SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 1-17, February 1991

VARIABLE METRIC METHOD FOR MINIMIZATION*

WILLIAM C. DAVIDON†

Abstract. This is a method for determining numerically local minima of differentiable functions of several variables. In the process of locating each minimum, a matrix which characterizes the behavior of the function about the minimum is determined. For a region in which the function depends quadratically on the variables, no more than N iterations are required, where N is the number of variables. By suitable choice of starting values, and without modification of the procedure, linear constraints can be imposed upon the variables.

Key words. variable metric algorithms, quasi-Newton, optimization

AMS(MOS) subject classifications. primary, 65K10; secondary, 49D37, 65K05, 90C30

A belated preface for ANL 5990. Enrico Fermi and Nicholas Metropolis used one of the first digital computers, the Los Alamos Maniac, to determine which values of certain theoretical parameters (phase shifts) best fit experimental data (scattering cross sections) [8]. They varied one theoretical parameter at a time by steps of the same magnitude, and when no such increase or decrease in any one parameter further improved the fit to the experimental data, they halved the step size and repeated the process until the steps were deemed sufficiently small. Their simple procedure was slow but sure, and several of us used it on the Avidac computer at the Argonne National Laboratory for adjusting six theoretical parameters to fit the pion-proton scattering data we had gathered using the University of Chicago synchrocyclotron [9]. To see how accurately the six parameters were determined, I varied them from their optimum values, and used the resulting degradations in the fit to estimate a six-by-six error matrix. This matrix approximates the inverse of a Hessian matrix of second derivatives of the objective function f, and specifies a metric in the space of gradients ∇f. Conjugate displacements in the domain of a quadratic objective function change gradients by amounts which are orthogonal with respect to this metric. The key ideas that led me to the development of variable-metric algorithms were 1) to update a metric in the space of gradients during the search for an optimum, rather than waiting until the search was over, and 2) to accelerate convergence by using each updated metric to choose the next search direction. In those days, we needed faster convergence to get results in the few hours between expected failures of the Avidac's large roomful of a few thousand bytes of temperamental electrostatic memory.

Shortly after joining the theoretical physics group at Argonne National Laboratory in 1956, I programmed the first variable-metric algorithms for the Avidac and used them to analyze the scattering of pi mesons by protons [10]. In 1957, I submitted a brief article about these algorithms to the Journal of Mathematics and Physics. This article was rejected, partly because it lacked proofs of convergence. The referee also found my notation "a bit bizarre," since I used "+" rather than "$k+1$" to denote updated quantities, as in "$x_+ = x + \alpha s$" rather than "$x_{k+1} = x_k + \alpha_k s_k$." While I then turned to other research, another member of our theoretical physics group, Murray Peshkin, modified and adapted one of these programs for Argonne's IBM 650. An

* This belated preface was received by the editors June 4, 1990; accepted for publication August 10, 1990. The rest of this article was originally published as Argonne National Laboratory Research and Development Report 5990, May 1959 (revised November 1959). This work was supported in part by University of Chicago contract W-31-109-eng-38.

† Department of Mathematics, Haverford College, Haverford, Pennsylvania 19041-1392.

Argonne physicist who used the program, Gilbert Perlow, urged me to publish a description of the algorithm so that he and Andrew Stehney could refer to it in a paper they were writing about their analysis of the radioactive decay of certain fission products [7]. Their reference thirteen was the first one to the report I was preparing [ANL-5990].

While this report focused mainly on the particular variable-metric algorithm which seemed to work best, it divided all algorithms of this type into five parts:

1. Choose a step direction s by acting on the current gradient $\mathbf{g}$ with the current metric in gradient space. If in this metric, $\mathbf{g}$ has a sufficiently small magnitude, then go to step 5.

2. Estimate the location of an optimum in the direction $\mathbf{s}$; e.g., by making a cubic interpolation. Go to this location if a sufficient change in the objective function is expected, else choose a new direction.

3. Evaluate the objective function f **and and its gradient** ∇f at the location x chosen in step 2 and estimate the directional derivative $\partial \nabla f(x+\alpha s)/\partial \alpha|_{\alpha=0}$ of ∇f at x.

4. Update the metric in gradient space, so that it yields $\mathbf{s}$ when acting on the directional derivative estimated in step 3. Return to step 1.

5. Test the current metric and minimizer. If these seem adequate, then quit, else return to step 1.

The hunting metaphor used in the report to name these five parts was chosen with tongue in cheek, since I expected the report would be read mostly by friends who knew I opposed killing for sport. The report would have been clearer had it first presented just the basic algorithm, with only those features needed to optimize quadratic objective functions, without the various "bells and whistles," which were added to accelerate convergence for certain nonquadratic objective functions; e.g., Formula 6.1 for the components $g_{\mu s}(x) = \lambda \partial g_\mu(x+\alpha s)/\partial \alpha|_{\alpha=0}$ of the directional derivative of the gradient would simplify to $g_\mu^+ - g_\mu \to g_{\mu s}$. It would also have been clearer if the rank-two update to the metric presented in the body of the report (later known as the Davidon–Fletcher–Powell (DFP) update) had been compared with the symmetric rank-one update (which was relegated to the appendix because it had not worked as well on certain problems).

Optimization algorithms were not among my research interests for several years after writing ANL-5990, and I returned to them only after others had called my attention to Fletcher and Powell's pioneering work on the subject [11].

1. Introduction. The solution to many different types of physical and mathematical problems can be obtained by minimizing a function of a finite number of variables. Among these problems are least-squares fitting of experimental data, determination of scattering amplitudes and energy eigenvalues by variational methods, the solution of differential equations, etc. With the use of high-speed digital computers, numerical methods for finding the minima of functions have received increased attention. Some of the procedures which have been used are those of optimum gradient [1], conjugate gradients [2], the Newton–Raphson iteration (see, e.g., [3], [4]) and one by Garwin and Reich [5]. In many instances, however, all of these methods require a large number of iterations to achieve a given accuracy in locating the minimum. Also, for some behaviors of the function being minimized, the procedures do not converge.

The method presented in this paper has been developed to improve the speed and accuracy with which the minima of functions can be evaluated numerically. In addition, a matrix characterizing the behavior of the function in the neighborhood of the minimum is determined in the process. Linear constraints can be imposed upon the variables by suitable choice of initial conditions, without alteration of the procedure.

2. Notation. We will employ the summation convention:

$$a^{\mu}b_{\mu} \equiv \sum_{\mu=1}^{N} a^{\mu}b_{\mu}.$$

In describing the iterative procedure, we will use symbols for memory locations rather than successive values of a number; e.g., we would write $x+3 \to x$ instead of $x_i+3=x_{i+1}$. In this notation, the sequence of operations is generally relevant. The following symbols will be used.

x^{μ}: $\mu = 1, \cdots, N$: the set of N independent variables.

$f(\mathbf{x})$: the value of the function to be minimized evaluated at the point $\mathbf{x}$.

$g_{\mu}(\mathbf{x})$: the derivatives of $f(\mathbf{x})$ with respect to x^{μ} evaluated at x:

$$g_{\mu}(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x^{\mu}}.$$

$h^{\mu\nu}$: a nonnegative symmetric matrix which will be used as a metric in the space of the variables.

Δ: The determinant of $h^{\mu\nu}$.

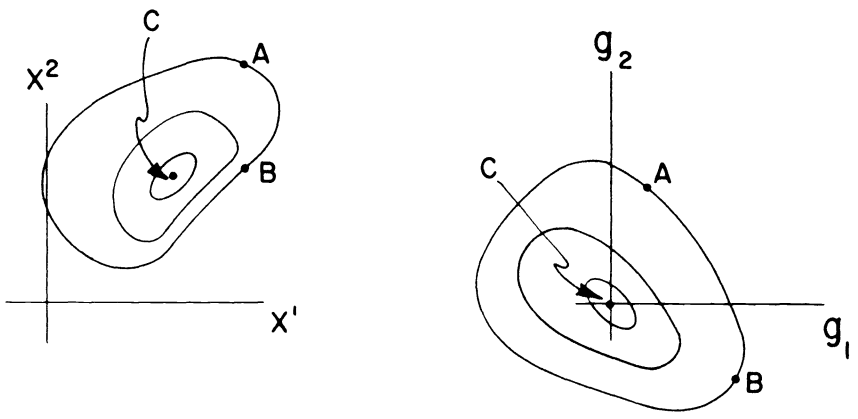
ε: 2 times fractional accuracy to which the function $f(\mathbf{x})$ is to be minimized.

d: a limiting value for what is to be considered as a "reasonable" minimum value of the function. For least-squares problems, d can be set equal to zero.

K: an integer which specifies the number of times the variables are to be changed in a random manner to test the reliability of the determination of the minimum.

3. Geometrical interpretation. It is convenient to use geometrical concepts to describe the minimization procedure. We do so by considering the variables x^{μ} to be the coordinates of a point in an N-dimensional linear space. As shown in Fig. 1(a), the set of $\mathbf{x}$ for which $f(\mathbf{x})$ is constant forms an $N-1$ dimensional surface in this space. One of this family of surfaces passes through each $\mathbf{x}$, and the surface about a point is characterized by the gradient of the function at that point:

$$g_{\mu}(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x^{\mu}}.$$



(a) (b)

FIG. 1. *Geometrical interpretation of* x^{μ} *and* $g_{\mu}(\mathbf{x})$.

These N components of the gradient can in turn be considered as the coordinates of a point in a different space, as shown in Fig. 1(b). As long as $f(\mathbf{x})$ is differentiable at all points, there is a unique point $\mathbf{g}$ in the gradient space associated with each point $\mathbf{x}$ in the position space, though there may be more than one $\mathbf{x}$ with the same $\mathbf{g}$.

In the neighborhood of any one point A the second derivatives of $f(\mathbf{x})$ specify a linear mapping of changes in position, $\mathbf{dx}$, onto changes in gradient $\mathbf{dg}$, in accordance with the equation

$$dg_\mu = \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \, dx^\nu. \tag{3.1}$$

The vectors $\mathbf{dx}$ and $\mathbf{dg}$ will be in the same direction only if $\mathbf{dx}$ is an eigenvector of the Hessian matrix:

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|.$$

If the ratios among the corresponding eigenvalues are large, then for most $\mathbf{dx}$ there will be considerable difference in the directions of these two vectors.

All iterative gradient methods, of which this is one, for locating the minima of functions consist of calculating $\mathbf{g}$ for various $\mathbf{x}$ in an effort to locate those values of $\mathbf{x}$ for which $\mathbf{g} = 0$, and for which the Hessian matrix is positive definite. If this matrix were constant and explicitly known, then the value of the gradient at one point would suffice to determine the minimum. In that case the change desired in $\mathbf{g}$ would be $-\mathbf{g}$, so we would have

$$-g_\mu = \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \, \Delta x^\nu, \tag{3.2}$$

from which we could obtain Δx^ν by multiplying (3.2) by the inverse of the matrix

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|.$$

However, in most situations of interest,

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|$$

is not constant, nor would explicit evaluation at points that might be far from a minimum represent the best expenditure of time.

Instead, an initial trial value is assumed for the matrix,

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|^{-1}.$$

This matrix, denoted by $h^{\mu\nu}$, specifies a linear mapping of all changes in the gradient onto changes in position. It is to be symmetric and nonnegative (positive definite if there are no constraints on the variables). After making a change in the variable $\mathbf{x}$, this trial value is improved on the basis of the actual relation between the changes in $\mathbf{g}$ and $\mathbf{x}$. If

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|$$

is constant, then, after N iterations, not only will the minimum of the function be determined, but also the final value of $h^{\mu\nu}$ will equal

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|^{-1}.$$

We shall subsequently discuss the significance of this matrix in specifying the accuracy to which the variables have been determined.

The matrix $h^{\mu\nu}$ can be used to associate a squared length to any gradient, defined by $h^{\mu\nu}g_\mu g_\nu$. If the Hessian matrix were constant and $h^{\mu\nu}$ were its inverse, then $\frac{1}{2}h^{\mu\nu}g_\mu g_\nu$ would be the amount by which $f(\mathbf{x})$ would exceed its minimum value. We therefore consider $h^{\mu\nu}$ as specifying a metric, and when we refer to the lengths of vectors, we will imply their lengths using $h^{\mu\nu}$ as the metric. We have called the method a "variable metric" method to reflect the fact that $h^{\mu\nu}$ is changed after each iteration.

We have divided the procedure into five parts which, to a large extent, are logically distinct. This not only facilitates the presentation and analysis of the method, but it is convenient in programming the method for machine computation.

4. Ready: Chart 1. The function of this section is to establish a direction along which to search for a relative minimum, and to box off an interval in this direction within which a relative minimum is located. In addition, the criterion for terminating the iterative procedure is evaluated.

Those operations which are only performed at the beginning of the calculation and not repeated on successive iterations have been included in Chart 1. These include the loading of input data, initial printouts, and the initial calculation of the function and its gradient. This latter calculation is treated as an independent subroutine, which may on its initial and final calculations include some operations not part of the usual iteration, such as loading operations, calculation of quantities for repeated use, special printouts, etc. A counter recording the number of iterations has been found to be a convenience, and is labeled I.

The iterative part of the computation begins with "READY 1." The direction of the first step is chosen by using the metric $h^{\mu\nu}$ in the relation

$$-h^{\mu\nu}g_\nu \to s^\mu. \tag{4.1}$$

The "component" of the gradient in this direction is evaluated through the relation

$$s^\mu g_\mu \to g_s. \tag{4.2}$$

From (4.1) and (4.2) we see that $-g_s$ is the squared length of $\mathbf{g}$, and hence the improvement to be expected in the function is $-\frac{1}{2}g_s$. The positive definiteness of $h^{\mu\nu}$ insures that g_s is negative, so that the step is in a direction which (at least initially) decreases the function. If its decrease is within the accuracy desired, i.e., if $g_s + \varepsilon > 0$, then the minimum has been determined. If not, we continue with the procedure.

In a first effort to box in the minimum, we take a step which is twice the size that would locate the minimum if the trial $h^{\mu\nu}$ were

$$\left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|^{-1}.$$

However, in order to prevent this step from being unreasonably large when the trial $h^{\mu\nu}$ is a poor estimate, we restrict the step to a length such that $(\lambda s^\mu)g_\mu$, the decrease in the function if it continued to decrease linearly, is not greater than some preassigned maximum, $2(f-d)$. We then change x^μ by

$$x^\mu + \lambda s^\mu \to x^{+\mu}, \tag{4.3}$$

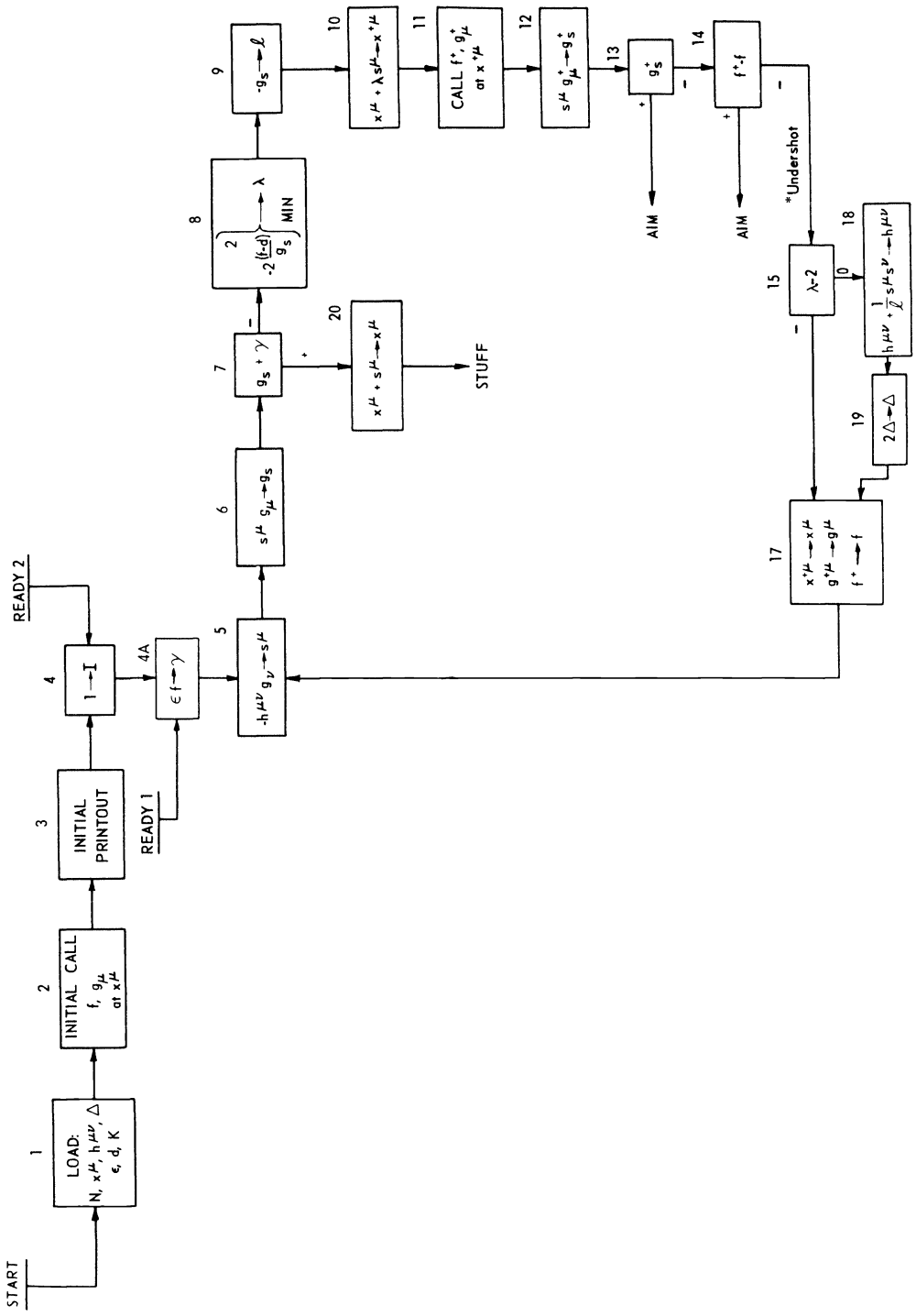


CHART 1: *Ready.*

* Optionally Printed Statements

and calculate the new value of the function and its gradient at $x^{+\mu}$. If the projection $s^\mu g_\mu^+ = g_s^+$ of the new gradient in the direction of the step is positive, or if the new value of the function f^+ is greater than the original f, then there is a relative minimum along the direction s between $\mathbf{x}$ and $\mathbf{x}^+$, and we proceed to "Aim" where we will interpolate its position. However, if neither of these conditions is fulfilled, the function has decreased and is decreasing at the point $\mathbf{x}^+$, and we infer that the step taken was too small. If the step had been limited by the preassigned change in the function d, we double d. If the step had been taken on the basis of $h^{\mu\nu}$, we modify $h^{\mu\nu}$ so as to double the squared length of s^μ, leaving the length of all perpendicular vectors unchanged. This is accomplished by

$$h^{\mu\nu} + \frac{1}{\ell} s^\mu s^\nu \to h^{\mu\nu}, \tag{4.4}$$

where ℓ is the squared length of s^μ. This doubles the determinant of $h^{\mu\nu}$. The process is then repeated, starting from the new position.

5. Aim: Chart 2. The function of this section is to estimate the location of the relative minimum within the interval selected by "Ready." Also, a comparison is made of the improvement expected by going to this minimum with that from a step perpendicular to this direction.

Inasmuch as the interpolation is along a one-dimensional interval, it is convenient to plot the function along this direction as a simple graph (see Fig. 2).

The values of f and f^+ of the function at points $\mathbf{x}$ and $\mathbf{x}^+$ are known, and so are its slopes, g_s and g_s^+, at these two points. We interpolate for the location of the minimum by choosing the "smoothest" curve satisfying the obundary conditions at $\mathbf{x}$ and $\mathbf{x}^+$, namely, the curve defined as the one which minimizes

$$\int_0^\lambda d\alpha \left(\frac{d^2 f}{d\alpha^2}\right)^2$$

over the curve. This is the curve formed by a flat spring fitted to the known ordinates and slopes at the end points, provided the slope is small. The resulting curve is a cubic, and its slope at any α $(0 \leqq \alpha \leqq \lambda)$ is given by

$$g_s(\alpha) = g_s - \frac{2\alpha}{\lambda}(g_s + z) + \frac{\alpha^2}{\lambda^2}(g_s + g_s^+ + 2z), \tag{5.1}$$

where

$$z = \frac{3(f - f^+)}{\lambda} + g_s + g_s^+.$$

The root of (5.1) that corresponds to a minimum lies between 0 and 1 by virtue of the fact that $g_s < 0$ and either $g_s^+ > 0$ or $z < g_s + g_s^+$. It can be expressed as

$$\alpha_{\min} = \lambda(1 - a),$$

where

$$a = \frac{g_s^+ + \mathcal{Q} - z}{g_s^+ - g_s + 2\mathcal{Q}} \tag{5.2}$$

and

$$\mathcal{Q} = (z^2 - g_s g_s^+)^{1/2}.$$

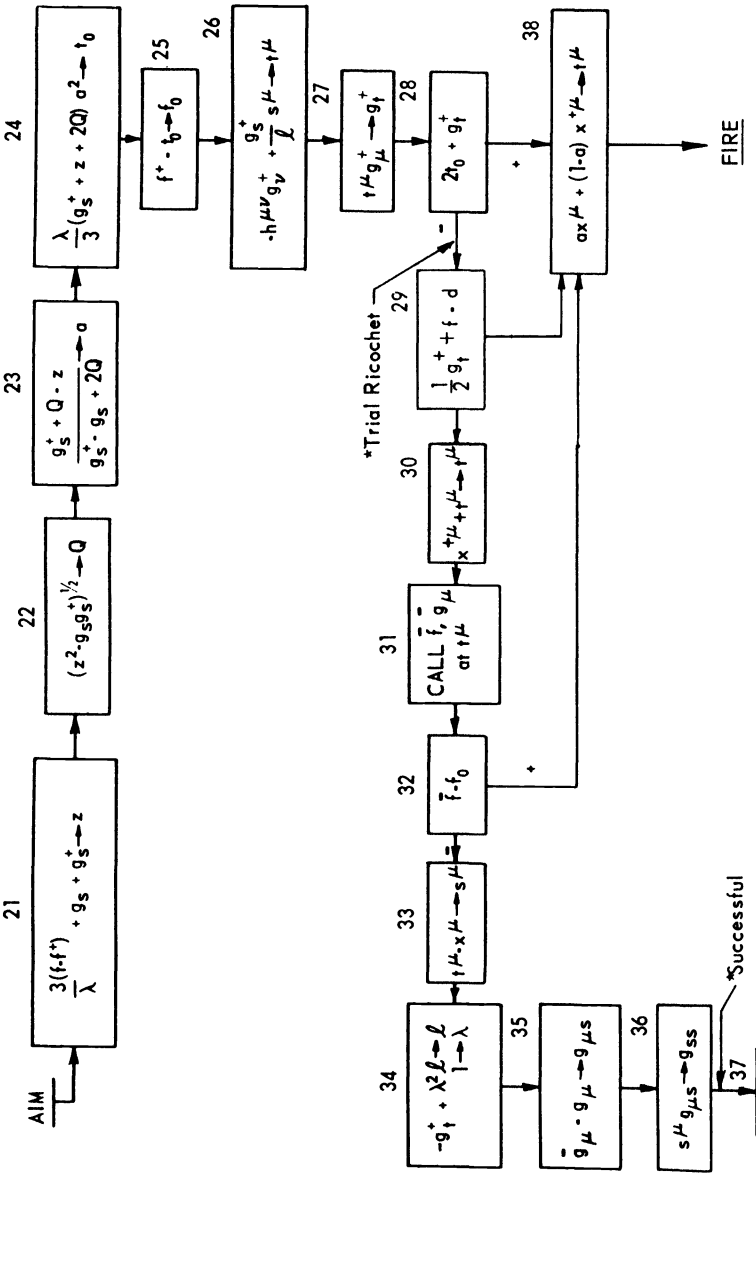

AIM
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
*Trial Ricochet
*Successful
DRESS 2
DRESS 1
FIRE

CHART 2: *Aim.*

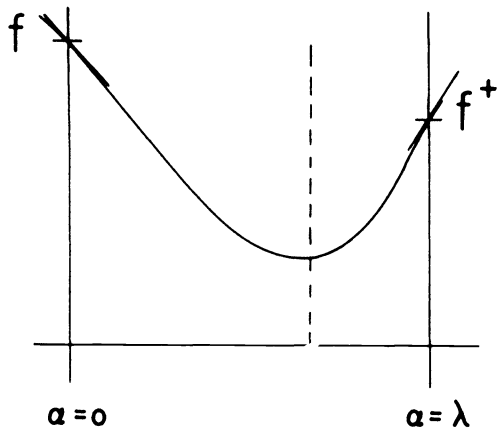


FIG. 2. *Plot of $f(\mathbf{x})$ along a one-dimensional interval.*

The particular form of (5.2) is chosen to obtain maximum accuracy, which might otherwise be lost in taking the difference of nearly equal quantities. The amount by which the minimum in f is expected to fall below f^+ is given by

$$\int_{(\lambda-\mathbf{a}\lambda)}^{\lambda} d\alpha\, g_s(\alpha) = \frac{1}{3}(g_s^+ + z + 2\mathfrak{Q})\, a^2\lambda. \tag{5.3}$$

The anticipated change is now compared with what would be expected from a perpendicular step. On the basis of the metric $h^{\mu\nu}$, the step to the optimum point in the $(N-1)$-dimensional surface perpendicular to s^μ through $x^{+\mu}$ is given by

$$-h^{\mu\nu}g_\nu^+ + \frac{g_s^+}{\ell}\, s^\mu \to t^\mu. \tag{5.4}$$

The change in f to be expected from this step is $\frac{1}{2}t^\mu g_\mu^+$. Hence, the decision whether to interpolate for the minimum along **s** or to change **x** by use of (5.4) is made by comparing $g_t^+ = t^\mu g_\mu^+$ with expression (5.3).

The purpose of allowing for this option is to improve the speed of convergence when the function is not quadratic. Consider the situation of Fig. 3. The optimum point between **x** and $\mathbf{x}^+$ is point *A*. However, by going to point *B*, a greater improvement can be made in the function. When the behavior of the function is described by a curving valley, this option is of particular value, for it enables successive iterations to proceed around the curve without backtracking to the local minimum along each step. However, if evaluation of the function at this new position does not give a better value than that expected from the interpolation, then the interpolated position is used. Should the new position be better as expected, it is then desired to modify $h^{\mu\nu}$ to incorporate the new information obtained about the function. The full step taken is stored at s^μ, and its squared length is the sum of the squares of the step along **s** and the perpendicular step, i.e., $s^\mu = -g_t^+ + \lambda^2\ell$. The change in the gradient resulting from this step is stored at $g_{\mu s}$ and these quantities are used in § 7 in a manner to be described.

For the interpolated step, we set

$$ax^\mu + (1-a)x^{+\mu} \to t^\mu. \tag{5.5}$$

By direct use of the x^μ instead of the s^μ, greater accuracy is obtained in the event that a is small. After making this interpolation, we proceed to "Fire."

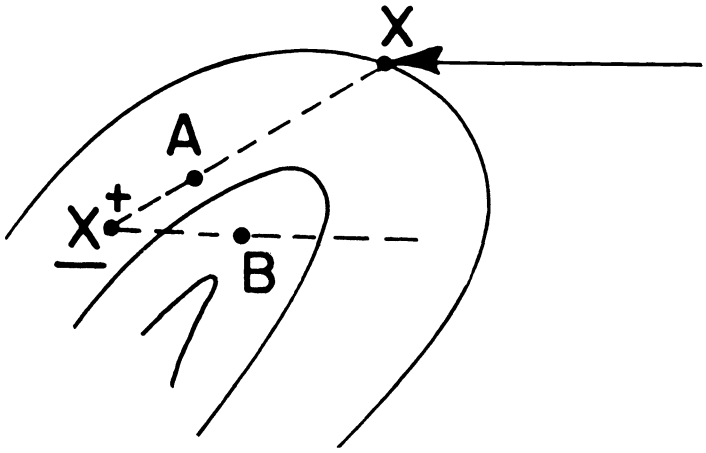


FIG. 3. *Illustration of procedure for nonquadratic functions. Point A is the optimum point along* (x, x^+); *point B is the location for the new trial.*

6. Fire: Chart 3. The purposes of this section are to evaluate the function and its gradient at the interpolated point and to determine if the local minimum has been sufficiently well located. If so, then the rate of change of gradient is evaluated (or, more accurately, λ times the rate of change) by interpolating from its values at $\mathbf{x}$, $\mathbf{x}^+$, and at the interpolated point.

If the function were cubic, then f at the interpolated point would be a minimum, the component of the gradient at this point along $\mathbf{s}$ would be zero, and the second derivative of the function at the minimum along the line would be $2\mathcal{Q}/\lambda$. However, as the function will generally be more complicated, none of these properties of f and its derivatives at the interpolated point will be exactly satisfied. We designate the actual value of f and its gradient at the interpolated point by $\bar{f}$ and $\bar{g}_\mu$. The component of $\bar{g}_\mu$ along s is $s^\mu \bar{g}_\mu = \bar{g}_s$. Should $\bar{f}$ be greater than f or f^+ by a significant amount $(>\varepsilon)$, the interpolation is not considered satisfactory and a new one is made within that part of the original interval for which f at the end point is smaller.

From the values of the gradient g_μ, $\bar{g}_\mu$, and g_μ^+ at three points along a line, we can interpolate to obtain its rate of change at the interpolated point. With a quadratic interpolation for the gradient, we obtain

$$(\bar{g}_\mu - g_\mu)\frac{a}{1-a} + (g_\mu^+ - \bar{g}_\mu)\frac{1-a}{a} \to g_{\mu s}, \tag{6.1}$$

where $g_{\mu s}/\lambda$ is the rate of change of the gradient at the interpolated point. The component of $g_{\mu s}$ in the direction of s, namely, $s^\mu g_{\mu s} = g_{ss}$, can be expressed as

$$\bar{g}_s\left(\frac{a}{1-a} - \frac{1-a}{a}\right) + 2\mathcal{Q} \to g_{ss}. \tag{6.2}$$

If the interpolated point were a minimum, then $\bar{g}_s = 0$ and $g_{ss} = 2\mathcal{Q}$.

An additional criterion imposed upon the interpolation is that the first term on the left of (6.2) be smaller in magnitude than $\mathcal{Q}$. Among other things, this insures that the interpolated value for the second derivative is positive. If this criterion is not fulfilled, no interpolation is made, and the matrix $h^{\mu\nu}$ is changed in a less sophisticated manner.

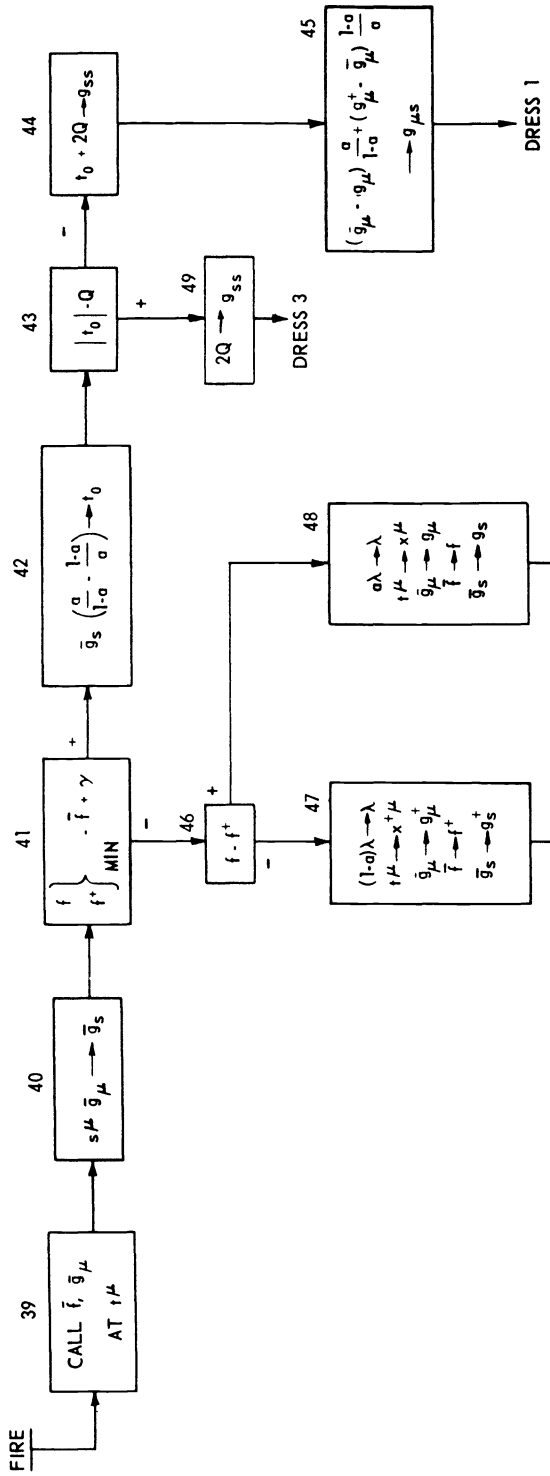

FIRE
39
CALL $\bar{f}$, $\bar{g}_\mu$ AT t^μ
40
s^μ $\bar{g}_\mu$ → $\bar{g}_s$
41
$\left.\begin{matrix} f \\ f^+ \end{matrix}\right\}$ MIN
$-\bar{f} + \gamma$
42
$\bar{g}_s \left(\frac{\alpha}{1-\alpha} \cdot \frac{1-\alpha}{\alpha}\right) \rightarrow t_0$
43
$|t_0| \cdot Q$
44
$t_0 + 2Q \rightarrow g_{ss}$
45
$(\bar{g}_\mu \cdot g_\mu) \frac{\alpha}{1-\alpha} + (g^+_\mu \cdot \bar{g}_\mu) \frac{1-\alpha}{\alpha}$
→ $g_{\mu s}$
DRESS 1
46
$f - f^+$
47
$(1-\alpha)\lambda \rightarrow \lambda$
$t^\mu \rightarrow x^{+\mu}$
$\bar{g}_\mu \rightarrow g^+_\mu$
$\bar{f} \rightarrow f^+$
$\bar{g}_s \rightarrow g^+_s$
AIM
48
$\alpha\lambda \rightarrow \lambda$
$t^\mu \rightarrow x^\mu$
$\bar{g}_\mu \rightarrow g_\mu$
$\bar{f} \rightarrow f$
$\bar{g}_s \rightarrow g_s$
AIM
49
$2Q \rightarrow g_{ss}$
DRESS 3

CHART 3: *Fire.*

7. Dress: Chart 4. The purpose of this section is to modify the metric $h^{\mu\nu}$ on the basis of information obtained about the function along the direction **s**. The new $h^{\mu\nu}$ is to have the property that $(h^{\mu\nu})'g_{\nu s} = \lambda s^{\mu}$, and must retain the information which the preceding iterations had given about the function.

If the vector $h^{\mu\nu}g_{\nu s} = t^{\mu}$ were in the direction of $\mathbf{s}^{\mu}$, then it would be sufficient to add to $h^{\mu\nu}$ a matrix proportional to $s^{\mu}s^{\nu}$. If t^{μ} is not in the direction of s^{μ}, the smallest squared length for the difference between s^{μ} and $(h^{\mu\nu} + \alpha s^{\mu}s^{\nu})g_{\nu s}$ is obtained when $\alpha = (\lambda / g_{ss}) - (1/\ell)$. For this value of α, the squared length of the difference is $t_0 - (g_{ss}/\ell)$ where t_0 is the square length of **d**, namely, $h^{\mu\nu}d_{\mu}d_{\nu}$. When this quantity is sufficiently small ($<\varepsilon$), the matrix $h^{\mu\nu}$ undergoes the change:

$$h^{\mu\nu} + \left(\frac{\lambda}{g_{ss}} - \frac{1}{\ell}\right) s^{\mu}s^{\nu} \to h^{\mu\nu}. \tag{7.1}$$

The corresponding change in the determinant of $h^{\mu\nu}$ is

$$\frac{\lambda \ell}{g_{ss}} \Delta \to \Delta. \tag{7.2}$$

When the vectors t^{μ} and s^{μ} are not sufficiently colinear, it is necessary to modify $h^{\mu\nu}$ by a matrix of rank two instead of one, i.e.,

$$h^{\mu\nu} - \frac{t^{\mu}t^{\nu}}{t_0} + \frac{\lambda}{g_{ss}} s^{\mu}s^{\nu} \to h^{\mu\nu}. \tag{7.3}$$

Then the change in the determinant of $h^{\mu\nu}$ is

$$\frac{\lambda g_{ss}}{t_0} \Delta \to \Delta. \tag{7.4}$$

After the matrix is changed, the iteration is complete; after printing out whatever information is desired about this part of the calculation, a new iteration is begun. This is repeated until the function is minimized to within the accuracy required.

8. Stuff: Chart 5. The purposes of this section are to test how well the function has been minimized and to test how well the matrix $h^{\mu\nu}$ approximates

$$\left\| \frac{\partial^2 f}{\partial x^{\mu}\, \partial x^{\nu}} \right\|$$

at the minimum. This is done by displacing point **x** from the location of the minimum in a random direction.

The displacement of point **x** is chosen to be a unit length in terms of $h^{\mu\nu}$ as the metric. When

$$h^{\mu\nu} = \left\| \frac{\partial^2 f}{\partial x^{\mu}\, \partial x^{\nu}} \right\|^{-1},$$

such a step will increase f by half the square of the length of the step.

If the direction were to be randomly distributed, then it would not be satisfactory to choose the range of each component of t_{μ} independently; rather, the range for the t_{μ} should be such that $h^{\mu\nu}t_{\mu}t_{\nu}$ is bounded by preassigned values. However, this refinement has not been incorporated into the charts nor the computer program. The length of the step has been chosen equal to one so that the function should increase by $\frac{1}{2}$ when each random step is taken.

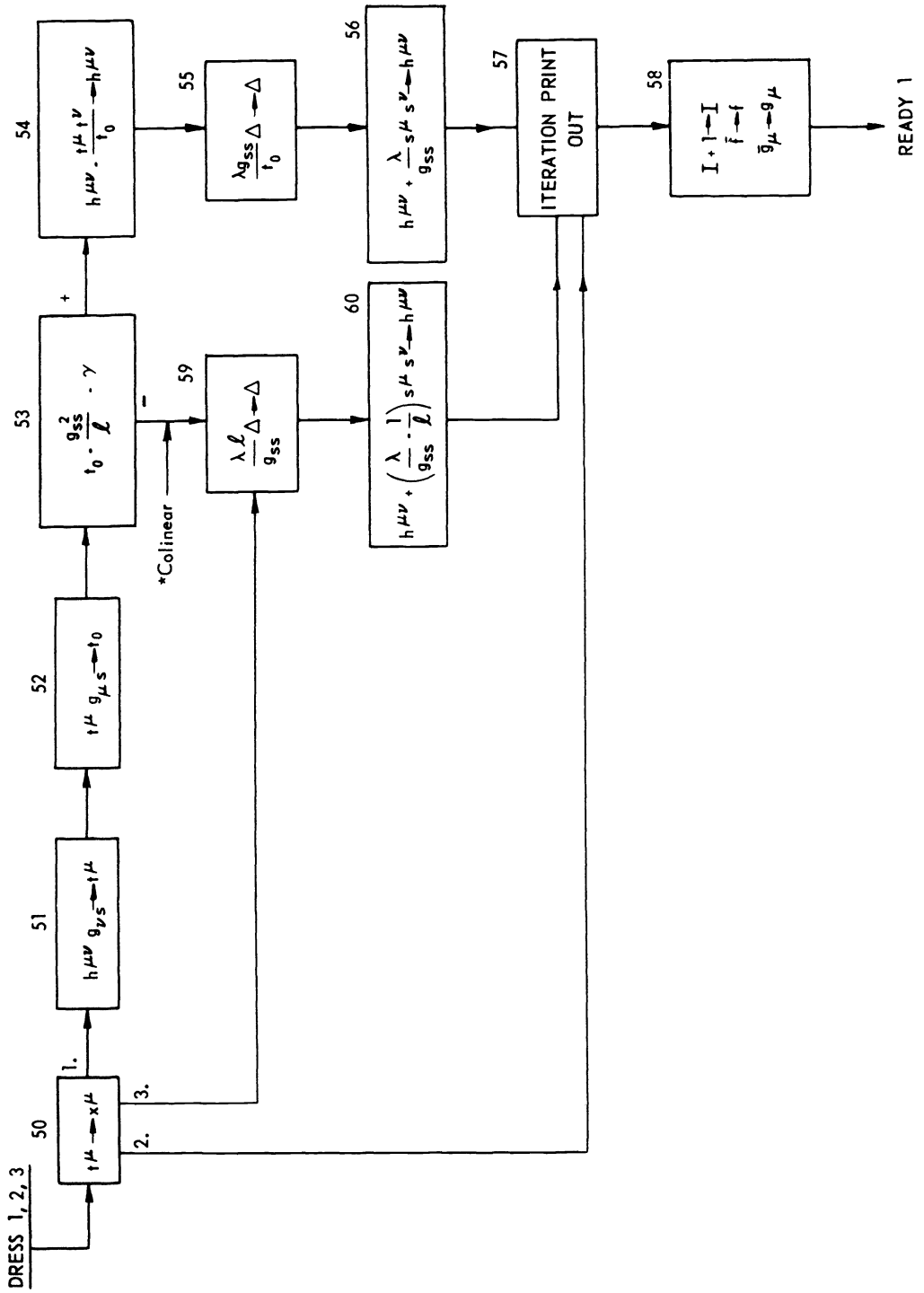


CHART 4: *Dress.*

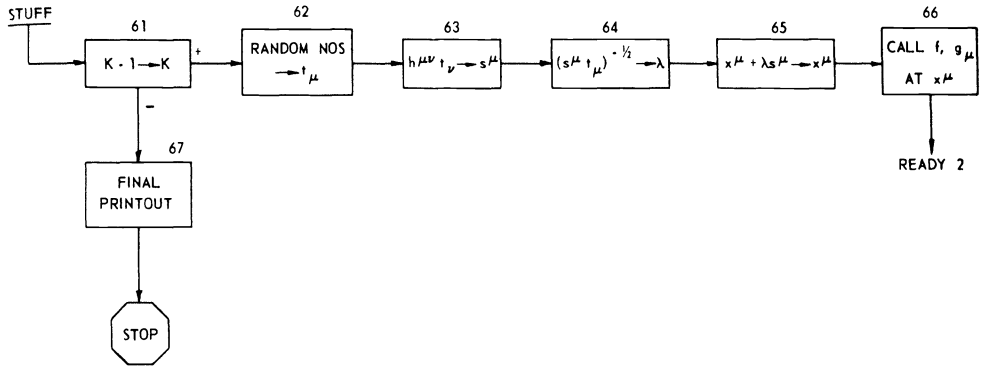


CHART 5: *Stuff.*

Significance of $h^{\mu\nu}$. We examine a least-squares analysis to illustrate how the initial trial value for $h^{\mu\nu}$ is chosen, and what its final value signifies. In this case, the function to be minimized will be chosen to be $\chi^2/2$, where χ^2 is the statistical measure of goodness of fit. The function $\chi^2/2$ is the natural logarithm of the relative probability for having obtained the observed set of data as a function of the variables χ^μ being determined.

The matrix

$$h^{\mu\nu} = \left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|^{-1}$$

then specifies the spreads and correlations among the variables by

$$\langle \Delta x^\mu \Delta x^\nu \rangle = \frac{\int d^N x (x^\mu - \langle x^\mu \rangle)(x^\nu - \langle x^\nu \rangle) \, e^{-\chi^2/2}}{\int d^N x \, e^{-\chi^2/2}} \approx h^{\mu\nu}. \tag{8.1}$$

The diagonal elements of $h^{\mu\nu}$ give the mean-square uncertainty for each of the variables, while the off-diagonal elements determine the correlations among them. The full significance of this matrix (the error matrix) is to be found in various works on statistics (see, for example, [6]). It enables us to determine the uncertainty in any linear function of the variables, for, if $u = a_\mu x^\mu$, then

$$\begin{aligned} \langle u \rangle &= a_\mu \langle x^\mu \rangle \\ \langle (\Delta u)^2 \rangle &= a_\mu a_\nu (\langle x^\mu x^\nu \rangle - \langle x^\mu \rangle \langle x^\nu \rangle) \\ &= a_\mu a_\nu h^{\mu\nu}. \end{aligned} \tag{8.2a}$$

If u is a more general function of $\mathbf{x}$, then if, in a Taylor expansion about the value of $\mathbf{x}$, derivatives higher than first can be ignored, we have

$$\begin{aligned} \langle u(\mathbf{x}) \rangle &= u(\langle \mathbf{x} \rangle) \\ \langle \Delta u(\mathbf{x}) \rangle^2 &= \frac{\partial u}{\partial x^\mu} (\langle \mathbf{x} \rangle) \frac{\partial u}{\partial x^\nu} (\langle \mathbf{x} \rangle) h^{\mu\nu}. \end{aligned} \tag{8.2b}$$

If it is possible to estimate the accuracy with which the variables are determined, the use of such estimates in the initial trial value of $h^{\mu\nu}$ will speed the convergence of the minimization procedure. Suppose, for example, that to fit some set of experi-

mental data, it is estimated that the variables x^μ have the values:

$$\begin{aligned} x^1 &= 3.0 \pm 0.1 \\ x^2 &= 28.0 \pm 2 \\ x^3 &= 10^4 \pm 10^2. \end{aligned} \tag{8.3}$$

Then, the initial values for x^μ and $h^{\mu\nu}$ would be

$$\begin{aligned} x^\mu &= (3.0 \quad 28.0 \quad 10^4) \\ h^{\mu\nu} &= \begin{pmatrix} 0.01 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 10^4 \end{pmatrix}. \end{aligned} \tag{8.4}$$

If this estimate is even correct to within a couple of orders of magnitude, the number of iterations required to locate the minimum may be substantially fewer than that for some more arbitrary choice, such as the unit matrix.

If it is desired to impose linear constraints on the variables, this can be readily done by starting with a matrix $h^{\mu\nu}$, which is no longer positive definite, but which has zero eigenvalues. For the constraints

$$\begin{aligned} a_\mu x^\mu &= \alpha \\ b_\mu x^\mu &= \beta, \end{aligned} \tag{8.5}$$

etc., the matrix $h^{\mu\nu}$ must be chosen so that

$$\begin{aligned} h^{\mu\nu} a_\nu &= 0 \\ h^{\mu\nu} b_\nu &= 0, \end{aligned} \tag{8.6}$$

and the starting value for x^μ must satisfy (8.5). For example, if x^3 is to be held constant, all elements of $h^{\mu\nu}$ in the third row and third column are set equal to zero and x^3 is set equal to the constant value.

When constraints are imposed instead of setting Δ equal to the determinant of $h^{\mu\nu}$ $(=0)$, it is set equal to the product of the nonzero eigenvalue of $h^{\mu\nu}$. Then, except for roundoff errors, not only will the conditions (8.6) be preserved in subsequent iterations, but also Δ will continue to equal the product of nonzero eigenvalues.

Though Δ is not used in the calculations, its value may be of interest in estimating how well the variables have been determined, since $\sum_\mu h^{\mu\mu}$ gives the sum of the eigenvalues of $h^{\mu\nu}$, while Δ gives their product. The square root of each of these eigenvalues is equal to one of the principal semiaxes of the ellipse formed by all $\mathbf{x}$ for which $f(\mathbf{x})$ exceeds its minimum value by $\frac{1}{2}$.

9. Conclusion. The minimization method described has been coded for the IBM-704 using Fortran. Experience is now being gathered on the operation of the method with diverse types of functions. Parts of the procedure, not incorporating all of the provisions described here, have been in use for some time in least-squares calculations for such computations as the analysis of $\pi - P$ scattering experiments [10], for the analysis of delayed neutron experiments [7], and similar computations. Though full mathematical analysis of its stability and convergence has not been made, general considerations and numerical experience with it indicate that minima of functions can be generally more quickly located than in alternate procedures. The ability of the

metric, $h^{\mu\nu}$, to accumulate information about the function and to compensate for ill-conditioned $g_{\mu\nu}$ is the primary reason for this advantage.

10. Acknowledgment. The author wishes to thank Dr. G. Perlow and Dr. M. Peshkin for valued discussions and suggestions, and Mr. K. Hillstrom for carrying out the computer programming and operation.

Appendix.[1] If we have the gradient of the function at a point in the neighborhood of a minimum together with $\mathbf{G}^{-1}$, where

$$\mathbf{G} = \left\| \frac{\partial^2 f}{\partial x^\mu \, \partial x^\nu} \right\|,$$

then, neglecting terms of higher order, the location of the minimum would be given in matrix notation by

$$\xi = x - \mathbf{G}^{-1}\nabla. \tag{1}$$

In the method to be described, a trial matrix is used for $\mathbf{G}^{-1}$ and a step determined by (1) is taken. From the change in the gradient resulting from this step, the trial value is improved and this procedure is repeated. The changes made in the trial value for $\mathbf{G}^{-1}$ are restricted to keep the hunting procedure "reasonable" regardless of the nature of the function. Let $\mathbf{H}$ be the trial value for $\mathbf{G}^{-1}$. Then the step taken will be to the point

$$x^+ = x - \mathbf{H}\nabla. \tag{2}$$

The gradient at x^+, ∇^+, is then evaluated. Let $D = \nabla^+ - \nabla$ be the change in the gradient as a result of the step $S = x^+ - x = -H\nabla$. We form the new trial matrix by

$$H^+_{\mu\nu} = H_{\mu\nu} + a(\mathbf{H}\nabla^+)_\mu(\mathbf{H}\nabla^+)_\nu. \tag{3}$$

The constant a is determined by the following two conditions:

1. The ratio of the determinant of $\mathbf{H}^+$ to that of $\mathbf{H}$ should be between R^{-1} and R, where R is a preassigned constant greater than 1. This is to prevent undue changes in the trial matrix and, in particular, if $\mathbf{H}$ is positive definite, $\mathbf{H}^+$ will be positive definite also.
2. The nonnegative quantity

$$\Delta = D\mathbf{H}^+D + S(\mathbf{H}^+)^{-1}S - 2S\cdot D \tag{4}$$

is to be minimized. This quantity vanishes when $S = \mathbf{H}^+D$. The a which satisfies these requirements, together with the corresponding Δ, as functions of $N = \nabla^+ H \nabla^+$ and $M = \nabla^+ H \nabla$, are as follows:[2]

(5)

Range of M	a	Δ
$M < -N/(R-1)$	$1/(M-N)$	0
$-N/(R-1) < M < N/(R+1)$	$(1/RN)-(1/N)$	$(N-M+MR)^2/RN$
$N/(R+1) < M < NR/(R+1)$	$(N-2M)/N(M-N)$	$4M(N-M)/N$
$NR/(R+1) < M < NR/(R-1)$	$(R/N)-(1/N)$	$(M+NR-MR)^2/RN$
$NR/(R-1) < M$	$1/(M-N)$	0

The dependence of Δ on M is bell-shaped, symmetric about a maximum at $M = N/2$, for which $a = 0$ and $\Delta = N$.

[1] The following method is a description of a simplified method embodying some of the ideas of the procedure presented in this report.

[2] When the function is known to be quadratic, the first condition can be dispersed with, in which case $a = (M-N)^{-1}$, $\Delta = 0$.

After forming the new trial matrix $\mathbf{H}^+$, the next step is taken in accordance with (2) and the process repeated, provided that $N = \nabla^+ H \nabla^+$ is greater than some preassigned ε. When the $\mathbf{G}$ is constant, Δ can be written as

$$\nabla = \mathbf{G}(x - \xi). \tag{6}$$

If u is an eigenvector of $\mathbf{HG}$ with eigenvalue one, then it will be an eigenvector of $\mathbf{H}^+\mathbf{G}$ with eigenvalue one as well, since

$$\begin{aligned} \mathbf{H}^+\mathbf{G}u &= \mathbf{HG}u + a\mathbf{H}\nabla^+(\nabla^+\mathbf{HG}u) \\ &= u + a\mathbf{H}\nabla^+[\nabla\mathbf{HG}(1-\mathbf{HG})u] \\ &= u. \end{aligned} \tag{7}$$

Furthermore, when $\Delta = 0$,

$$\mathbf{H}^+\mathbf{G}S = \mathbf{H}^+ D = S, \tag{8}$$

so that S becomes another such eigenvector. After no more than N steps (for which $\Delta = 0$), $\mathbf{H}$ will equal $\mathbf{G}^{-1}$ and the following step will be to the exact minimum.

The entire procedure is covariant under an arbitrary linear coordinate transformation. Under these transformations of x, ∇ transforms as a covariant vector, $\mathbf{G}$ transforms as a covariant tensor of second rank, and $\mathbf{H}$ transforms as a contravariant tensor of second rank. The intrinsic characteristics of a particular hunting calculation are determined by the eigenvalues of the mixed tensor $\mathbf{HG}$, and the components of the initial value of $(x - \xi)$ along the direction of the corresponding eigenvectors. Since successive steps will bring $\mathbf{HG}$ closer to unity, convergence will be rapidly accelerating even when $\mathbf{G}$ itself is ill-conditioned. Constraints of the form $b \cdot x = c$ can be improved by using an initial $\mathbf{H}$ which annuls b, i.e.,

$$\mathbf{H} \cdot b = 0,$$

and choosing the initial vector $\mathbf{x}$ such that it satisfies $b \cdot x = c$. Then all steps taken will be perpendicular to b and this inner product will be conserved. For example, if it is desired to hold one component of x constant, all the elements of $\mathbf{H}$ corresponding to that component are initially set equal to zero.

REFERENCES

[1] A. CAUCHY, *Méthode générale pour la résolution des systèmes d'équations simultanées*, Compt. Rend., 25, 536 (1847).

[2] M. R. HESTENES AND C. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409-436.

[3] F. B. HILDEBRAND, *Introduction to Numerical Analysis*, McGraw-Hill, New York, 1956.

[4] W. A. NIERENBERG, Report UCRL-3816, University of California Radiation Laboratory, Berkeley, CA, 1957.

[5] R. L. GARWIN AND H. A. REICH, *An efficient iterative least squares method* (to be published).

[6] H. CRAMER, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, 1946.

[7] G. J. PERLOW AND A. F. STEHNEY, *Halogen delayed-neutron activities*, Phys. Rev., 113 (1959), pp. 1269-1276.

[8] E. FERMI AND N. METROPOLIS, Los Alamos unclassified report LA-1492, Los Alamos National Laboratory, Los Alamos, NM, 1952.

[9] H. L. ANDERSON, W. C. DAVIDON, M. G. GLICKSMAN, AND U. E. KRUSE, *Scattering of positive pions by hydrogen at* 189 *MeV*, Phys. Rev., 100 (1955), pp. 279-287.

[10] H. L. ANDERSON AND W. C. DAVIDON, *Machine analysis of pion scattering by the maximum likelihood method*, Nuovo Cimento, 5 (1957), pp. 1238-1255.

[11] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization*, Comput. J., 6 (1963), pp. 163-168.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 18–21, February 1991

002

A NEW VARIATIONAL RESULT FOR QUASI-NEWTON FORMULAE*

R. FLETCHER†

Abstract. The recent measure function of Byrd and Nocedal [*SIAM J. Numer. Anal.*, 26 (1989), pp. 727–739] is considered and simple proofs of some of its properties are given. It is then shown that the BFGS and DFP formulae satisfy a least change property with respect to this new measure.

Key words. quasi-Newton method, BFGS formula, DFP formula

AMS(MOS) subject classifications. 65K, 90C

1. Introduction. Recently Byrd and Nocedal [2] introduced the measure function $\psi : \mathbb{R}^{n\times n} \to \mathbb{R}$ defined by

$$\psi(A) = \text{trace}\,(A) - f(A) \tag{1.1}$$

where $f(A)$ denotes the function

$$f(A) = \ln(\det A). \tag{1.2}$$

Byrd and Nocedal use this function to unify and extend certain convergence results for quasi-Newton methods. In this paper, simple proofs of some of the properties of these functions are given. These properties are then used in § 2 to give a new variational result for the BFGS and DFP updating formulae.

LEMMA 1.1. *$f(A)$ is a strictly concave function on the set of positive definite diagonal $n \times n$ matrices.*

Proof. Let $A = \text{diag}\,(a_i)$. Then $\nabla^2 f = \text{diag}\,(-1/a_i^2)$ and is negative definite since $a_i > 0$ for all i. Hence f is strictly concave. □

LEMMA 1.2. *$f(A)$ is a strictly concave function on the set of positive definite symmetric $n \times n$ matrices.*

Proof. Let $A \neq B$ be any two such matrices. Then there exist $n \times n$ matrices X and Λ, (X is nonsingular, $\Lambda = \text{diag}\,(\lambda_i)$) such that $X^T A X = \Lambda$ and $X^T B X = I$. Denote $C = (1-\theta)A + \theta B$, $\theta \in (0, 1)$. Then

$$X^T C X = (1-\theta) X^T A X + \theta X^T B X = (1-\theta)\Lambda + \theta I. \tag{1.3}$$

Also,

$$\begin{aligned} f(X^T A X) &= \ln\det(X^T A X) = \ln(\det^2 X \det A) \\ &= f(A) + \ln\det^2 X, \end{aligned} \tag{1.4}$$

and likewise

$$f(X^T B X) = f(B) + \ln\det^2 X \tag{1.5}$$

$$f(X^T C X) = f(C) + \ln\det^2 X. \tag{1.6}$$

Now $A \neq B \Leftrightarrow \Lambda \neq I$, so by Lemma 1.1 and Eq. (1.3) it follows for $\theta \in (0, 1)$ that

$$f(X^T C X) = f((1-\theta)\Lambda + \theta I) > (1-\theta) f(\Lambda) + \theta f(I) = (1-\theta) f(X^T A X) + \theta f(X^T B X).$$

* Received by the editors January 12, 1990; accepted for publication (in revised form) May 7, 1990.

† Department of Mathematical Sciences, University of Dundee, Dundee DD1 4HN, Scotland, United Kingdom.

Hence from (1.4)-(1.6),

$$f(C) > (1-\theta)f(A) + \theta f(B),$$

and so the lemma is established. □

Remark. Concavity of $f(A)$ is proved elsewhere, for example as a consequence of (1) [1, Chap 8.5]. The above method of proof is different and also shows strict concavity.

LEMMA 1.3. *$\psi(A)$ is a strictly convex function on the set of positive definite symmetric $n \times n$ matrices.*

Proof. This follows from Lemma 1.2 and the linearity of trace (A). □

LEMMA 1.4. *For nonsingular A the derivative of* det (A) *is given by* $d(\det A)/da_{ij} = [A^{-T}]_{ij} \det A$.

Proof. From the well-known identity $\det(I + uv^T) = 1 + v^T u$ it follows that

$$\det(A + \varepsilon e_i e_j^T) = \det(I + \varepsilon e_i e_j^T A^{-1}) \det A = (1 + \varepsilon (A^{-1})_{ji}) \det A.$$

Hence

$$\frac{d \det A}{da_{ij}} = \lim_{\varepsilon \to 0} \frac{\det(A + \varepsilon e_i e_j^T) - \det A}{\varepsilon} = (A^{-1})_{ji} \det A. \qquad \square$$

THEOREM 1.1. *$\psi(A)$ is globally and uniquely minimized by $A = I$ over the set of positive definite symmetric $n \times n$ matrices.*

Proof. Because A is nonsingular, ψ is continuously differentiable and so

$$\frac{d\psi}{da_{ij}} = I_{ij} - \frac{1}{\det A} \frac{d}{da_{ij}} \det A = (I - A^{-T})_{ij}, \tag{1.7}$$

using Lemma 1.4. Hence ψ is stationary when $A = I$ and the theorem follows by virtue of Lemma 1.3. □

Remark. It is also shown in [2] that $A = I$ is a global minimizer of $\psi(A)$.

2. A variational result. The BFGS updating formula

$$B^{(k+1)} = B^{(k)} - \frac{B^{(k)} \delta \delta^T B^{(k)}}{\delta^T B^{(k)} \delta} + \frac{\gamma \gamma^T}{\delta^T \gamma} \tag{2.1}$$

and the DFP updating formula

$$H^{(k+1)} = H^{(k)} - \frac{H^{(k)} \gamma \gamma^T H^{(k)}}{\gamma^T H^{(k)} \gamma} + \frac{\delta \delta^T}{\gamma^T \delta} \tag{2.2}$$

occupy a central role in unconstrained optimization. (Here δ and γ denote certain difference vectors occurring on iteration k of a quasi-Newton method, with $\delta^T \gamma > 0$. $B^{(k)}$ denotes the current Hessian approximation, and $H^{(k)}$ its inverse: see, for example, [3] for details.) A significant result due to Goldfarb [4] is that the correction in the BFGS or DFP formula satisfies a minimum property with respect to a function of the form $\|E\|_W^2 = \text{trace}(EWEW)$ (e.g., Theorem 3.3.2 and its corollary in [3]). The main result of this paper is to show that these formulae also satisfy a minimum property with respect to the measure function ψ of Byrd and Nocedal defined in (1.1).

THEOREM 2.1. *If $H^{(k)}$ is positive definite and $\delta^T \gamma > 0$, the variational problem*

$$\underset{B>0}{\text{minimize}} \quad \psi(H^{(k)1/2} B H^{(k)1/2}) \tag{2.3}$$

$$\text{subject to} \quad B^T = B \tag{2.4}$$

$$B\delta = \gamma \tag{2.5}$$

is solved uniquely by the matrix $B^{(k+1)}$ given by the BFGS *formula* (2.1).

Proof. The matrix product that forms the argument of ψ can be cyclically permuted so that

$$\begin{aligned}\psi(H^{(k)1/2}BH^{(k)1/2}) &= \text{trace}\,(H^{(k)}B) - \ln(\det H^{(k)}\det B)\\ &= \psi(H^{(k)}B) = \psi(BH^{(k)}).\end{aligned} \tag{2.6}$$

A constrained stationary point of the variational problem can be obtained by the method of Lagrange multipliers. A suitable Lagrangian function is

$$\begin{aligned}L(B,\Lambda,\lambda) &= \tfrac{1}{2}\,\psi(H^{(k)1/2}BH^{(k)1/2}) + \text{trace}\,(\Lambda^T(B^T-B)) + \lambda^T(B\delta-\gamma)\\ &= \tfrac{1}{2}(\text{trace}\,(H^{(k)}B) - \ln\det H^{(k)} - \ln\det B) + \text{trace}\,(\Lambda^T(B^T-B))\\ &\quad + \lambda^T(B\delta-\gamma)\end{aligned}$$

where Λ and λ are Lagrange multipliers for (2.4) and (2.5), respectively. To solve the first order conditions, it is necessary to find B, Λ, and λ to satisfy (2.4), (2.5), and the equations $\partial L/\partial B_{ij} = 0$. Using the identity $\partial B/\partial B_{ij} = e_i e_j^T$ and Lemma 1.4, it follows that

$$\begin{aligned}\frac{\partial L}{\partial B_{ij}} = 0 &= \tfrac{1}{2}(\text{trace}\,(H^{(k)}e_ie_j^T) - (B^{-1})_{ji}) + \text{trace}\,(\Lambda^T(e_je_i^T - e_ie_j^T)) + \lambda^T e_ie_j^T\delta\\ &= \tfrac{1}{2}((H^{(k)})_{ji} - (B^{-1})_{ji}) + \Lambda_{ji} - \Lambda_{ij} + (\lambda\delta^T)_{ij}.\end{aligned}$$

Transposing and adding, using the symmetry of $H^{(k)}$ and B, gives

$$H^{(k)} - B^{-1} + \lambda\delta^T + \delta\lambda^T = 0$$

or

$$B^{-1} = H^{(k)} + \lambda\delta^T + \delta\lambda^T, \tag{2.7}$$

which shows that the optimum matrix inverse involves a rank-2 correction of $H^{(k)}$. To determine λ, (2.7) is post-multiplied by γ. It then follows, using the equation $B^{-1}\gamma = \delta$ derived from (2.5), that

$$\delta = H^{(k)}\gamma + \lambda\delta^T\gamma + \delta\lambda^T\gamma,$$

and hence

$$\gamma^T\delta = \gamma^TH^{(k)}\gamma + \gamma^T\lambda\delta^T\gamma + \gamma^T\delta\lambda^T\gamma.$$

Rearranging this gives $\gamma^T\lambda = \frac{1}{2}(1 - \gamma^TH^{(k)}\gamma/\delta^T\gamma)$, and so

$$\lambda = (\delta - H^{(k)}\gamma - \tfrac{1}{2}\delta(1 - \gamma^TH^{(k)}\gamma/\delta^T\gamma))/\delta^T\gamma.$$

Substituting this expression into (2.7) gives the equation

$$B^{-1} = H^{(k)} - \frac{H^{(k)}\gamma\delta^T + \delta\gamma^TH^{(k)}}{\delta^T\gamma} + \frac{\delta\delta^T}{\delta^T\gamma}\left(1 + \frac{\gamma^TH^{(k)}\gamma}{\delta^T\gamma}\right).$$

It is a well-known consequence of the Sherman–Morrison formula (e.g., [3]) that there exists a corresponding rank-2 update for B, which is given by the right-hand side of (2.1). Moreover, the conditions of the theorem ensure that the resulting updated matrix B is positive definite (as in [3, Thm. 3.2.2]). This establishes that the BFGS formula satisfies first order conditions (including feasibility) for the variational problem. Finally, $\psi(H^{(k)1/2}BH^{(k)1/2})$ is seen to be a strictly convex function on $B > 0$ by virtue of (2.6) and Lemma 1.2, so it follows that the BFGS formula gives the unique solution of the variational problem. □

COROLLARY. *If* $B^{(k)}$ *is positive definite and* $\delta^T\gamma > 0$, *the variational problem*

$$\begin{aligned} &\underset{H>0}{\text{minimize}} \quad \psi(B^{(k)1/2}HB^{(k)1/2}) \\ &\text{subject to} \quad H^T = H \\ &\qquad\qquad H\gamma = \delta \end{aligned}$$

is solved uniquely by the matrix $H^{(k+1)}$ *given by the* DFP *formula* (2.2).

Proof. The result follows by replacing (B, H, δ, γ) by (H, B, γ, δ) throughout Theorem 2.1. □

REFERENCES

[1] R. BELLMAN, *Introduction to Matrix Analysis*, McGraw-Hill, New York, 1960.

[2] R. H. BYRD AND J. NOCEDAL, *A tool for the analysis of quasi-Newton methods with application to unconstrained minimization*, SIAM J. Numer. Anal., 26 (1989), pp. 727–739.

[3] R. FLETCHER, *Practical Methods of Optimization*, Second Edition, John Wiley, Chichester, 1987.

[4] D. GOLDFARB, *A family of variable metric methods derived by variational means*, Math. Comp., 24 (1970), pp. 23–26.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 22-29, February 1991

003

ON THE PERFORMANCE OF KARMARKAR'S ALGORITHM OVER A SEQUENCE OF ITERATIONS*

KURT M. ANSTREICHER†

Abstract. Karmarkar's projective algorithm for linear programming is considered with real arithmetic and exact linesearch of the potential function. It is shown that for every $n \geqq 3$ there is a linear program, with n variables, such that the algorithm obtains a potential reduction of about 1.3 on each iteration. For the same problems the algorithm requires $\Theta(\ln(n/\varepsilon))$ iterations to reduce the objective gap to a factor ε of its initial value. It is thus proved that in the worst case the convergence of Karmarkar's algorithm, with exact linesearch, cannot be independent of n, and moreover, potential reduction may be a poor indicator of algorithm performance.

Key words. linear programming, Karmarkar's algorithm, potential function

AMS(MOS) subject classification. 90C05

1. Introduction. The true worst-case complexity of Karmarkar's projective algorithm for linear programming is a problem of considerable theoretical interest. In his original paper [4] introducing the algorithm, Karmarkar showed that the potential function, a surrogate for the linear objective, can be reduced by a constant on each iteration. As a consequence, the gap reduction time for the algorithm, applied to a linear program with n variables, is no worse than $O(n)$. (We use the phrase "gap reduction time" to refer to the number of iterations required to reduce the objective gap (value minus optimal value) by a given, constant factor.) In practice, however, the gap reduction time seems to be virtually independent of n, or perhaps grows very slowly, like $\ln(n)$.

The worst-case performance on a single step of the algorithm has been completely characterized (see Anstreicher [1] and McDiarmid [5]). In fact, it is possible that on a single step, using exact linesearch of the potential function, the potential value is reduced by only about 0.72. It is also known (see [5]) that using an "optimal" fixed steplength could result in the algorithm reducing the potential function by only about 0.69 on every iteration. (However, if a linesearch were performed in McDiarmid's example of the latter, the algorithm would terminate with optimality in a single iteration.) The asymptotic behavior of a "short step" implementation of the algorithm was obtained by Asic, Kovacevic-Vujcic, and Radosavljevic-Nikolic [2]. Finally, it has been shown that the algorithm, using exact linesearch of the potential function, does not produce superlinear convergence of the objective gap (see Bayer and Lagarias [3]).

In this paper we consider Karmarkar's algorithm, applied using exact linesearch of the potential function. We show, by construction, that for every $n \geqq 3$ there is a linear program such that the algorithm reduces the potential value by only about 1.3 on each iteration. This shows that the worst-case gap reduction time for the algorithm, with exact linesearch, cannot be improved by showing that potential reduction somehow increases as the algorithm iterates. We also show that the actual gap reduction time, for the same linear programs, is $\Theta(\ln(n))$. In particular, we show that the number of

*Received by the editors December 26, 1989; accepted for publication (in revised form) June 20, 1990. This work was written while the author was a research fellow at the Center for Operations Research and Econometrics, Université Catholique de Louvain, Louvain-La-Neuve, Belgium. It was first presented at the Second Asilomar Workshop on Progress in Mathematical Programming, Monterey, CA, February 5-7, 1990.

† Department of Operations Research, Yale University, New Haven, Connecticut 06520.

iterations required to reduce the gap to a factor ε of its initial value is $\Theta(\ln(n/\varepsilon))$, compared to the bound of $O(n \ln(1/\varepsilon))$ implied by the constant potential reduction. The two results taken together demonstrate that potential reduction may be a very poor indicator of the actual convergence of the algorithm. Similar results hold for the algorithm implemented with a "fixed fraction to the boundary" step rule, for any fraction in $(\frac{2}{3}, 1)$.

2. Karmarkar's algorithm. Consider a linear program

$$\begin{aligned} \text{LP} \quad z^* = \min\ & c^\top x \\ & Ax = 0 \\ & x \in S, \end{aligned}$$

where A is an $m \times n$ matrix, and $S \subset \mathcal{R}^n$ is the simplex $S = \{x \geqq 0 | e^\top x = n\}$. (Throughout, we will use e to denote the vector in $\mathcal{R}^n$ with each component equal to 1.) We assume that $Ae = 0$, and that $z^* = 0$. In [4] it is shown that any standard form linear program may be converted into the form above by combining the original linear program with its dual.

We now describe Karmarkar's algorithm for LP. Let $k \geqq 0$, and let $x^k > 0$ be an iterate of the algorithm, feasible for LP $(x^0 = e)$. Let X_k be the diagonal matrix $X_k = \operatorname{diag}(x^k)$. Using the projective transformation $T(\cdot): S \to S$,

$$\begin{aligned} y = T(x) &= \left(\frac{n}{e^\top X_k^{-1} x}\right) X_k^{-1} x \\ x = T^{-1}(y) &= \left(\frac{n}{e^\top X_k y}\right) X_k y, \end{aligned} \tag{2.1}$$

we obtain the transformed problem

$$\begin{aligned} \overline{\text{LP}} \quad \min\ & \bar{c}^\top y \\ & \bar{A} y = 0 \\ & y \in S, \end{aligned}$$

where $\bar{A} = AX_k$, $\bar{c} = X_k c$. Note that $T(x^k) = e$, so a step starting at e in $\overline{\text{LP}}$ corresponds to a step starting at x^k in LP. Under the assumption that $z^* = 0$, the optimal objective value in $\overline{\text{LP}}$ is also 0.

Let $g \in \mathcal{R}^n$ be some direction satisfying $\bar{A}g = 0$, $\bar{c}^\top g > 0$. The algorithm makes a step of the form

$$y(\lambda) = e - \lambda \frac{g}{g_{\max}}, \tag{2.2}$$

where $0 \leqq \lambda \leqq 1$, and $g_{\max}$ is equal to the maximum component of g. (Similarly, $g_{\min}$ will be used to denote the minimum component of g.) Note that $z^* = 0$ and $\bar{c}^\top g > 0$ imply that $g_{\max} > 0$, so λ is simply the fraction of a step to the boundary of the nonnegative orthant. In practice λ is often determined by an approximate linesearch of the potential function

$$f(\bar{c}, y) = n \ln(\bar{c}^\top y) - \sum_{j=1}^{n} \ln(y_j), \tag{2.3}$$

along $y = y(\lambda)$. It can be shown that if $y = T(x)$, then $f(\bar{c}, y)$ differs from $f(c, x)$ by a constant which is independent of x. Following the choice of λ, we set $x^{k+1} = T^{-1}(y(\lambda))$, and go to the next iteration.

Karmarkar's algorithm is usually described with $g = \bar{c}_p$, the projection of $\bar{c}$ onto $\{x \in \mathcal{R}^n \mid \bar{A}x = 0,\ e^\top x = 0\}$. For this choice of g it can be shown (see [1] or [5]) that

$$\frac{\bar{c}^\top y(\lambda)}{\bar{c}^\top e} \leqq 1 + \frac{\lambda \|g\|^2}{n g_{\min} g_{\max}} \leqq 1 - \frac{2\lambda}{n}, \tag{2.4}$$

and furthermore there is a λ such that $f(\bar{c}, y(\lambda)) - f(\bar{c}, e) < -.72$. It follows that the potential function $f(c, \cdot)$ can be reduced by at least 0.72 on each iteration, and such uniform reduction is sufficient to obtain a polynomial-time bound for the algorithm. It turns out, however, that when basing steplength on a linesearch of $f(\bar{c}, \cdot)$, it is equivalent to use $g = \bar{c}_q$, the projection of $\bar{c}$ onto the nullspace of $\bar{A}$. This was first noted by Jean-Philippe Vial (Todd [7]), and follows from the homogeneity of $f(\bar{c}, \cdot)$ and the fact that $\bar{A}e = 0$. Note that the inverse transformation in (2.1) may be applied to a point $0 \neq y \geqq 0$, to obtain an $x \in S$, even when $e^\top y \neq n$. In the sequel, we will find it convenient to describe the algorithm using $g = \bar{c}_q$.

Let $\bar{c}^k = X_k c$ denote the value of $\bar{c}$ on iteration k. Note that since $x^{k+1} = T^{-1}(y(\lambda))$, we have

$$\bar{c}^{k+1} = X_{k+1} c = \left(\frac{n}{e^\top X_k y(\lambda)} \right) X_k Y(\lambda) c \sim Y(\lambda) \bar{c}^k, \tag{2.5}$$

where $Y(\lambda)$ is the diagonal matrix diag $(y(\lambda))$, and $u \sim v$ indicates that two vectors u and v in $\mathcal{R}^n$ differ by a positive scaling.

3. A specific example. We now consider a simple, specific case:

$$\text{LP} \quad \min c^\top x$$
$$x \in S,$$

where $n \geqq 3$, $c = (0, 1, \cdots, 1, \gamma, \beta)^\top$, $0 < \gamma < 1$, $\beta > 1$. So $m = 0$, and we may take $g = \bar{c}_q = \bar{c}$ on each iteration. (Directions of a similar form have previously arisen in the analysis of Karmarkar's algorithm. In particular, the case $\gamma = 1$, $\beta = 2$ leads to $g = \bar{c}_p$ giving the worst case in (2.4), see [1], [5].) Consider the first iteration ($k = 0$), with $\bar{c} = c$. The step $y(\lambda)$, as in (2.2), is then

$$y(\lambda) = \left(1, 1 - \frac{\lambda}{\beta}, \cdots, 1 - \frac{\lambda}{\beta}, 1 - \frac{\lambda\gamma}{\beta}, 1 - \lambda \right)^\top. \tag{3.1}$$

Following the choice of λ, by (2.5) the rescaled objective $\bar{c}$ will be

$$\bar{c} \sim Y(\lambda) c = \left(0, \frac{\beta - \lambda}{\beta}, \cdots, \frac{\beta - \lambda}{\beta}, \frac{\gamma(\beta - \lambda\gamma)}{\beta}, \beta(1 - \lambda) \right)^\top.$$

Thus $\bar{c} \sim (0, 1, \cdots, 1, \bar{\beta}, \bar{\gamma})^\top$, where

$$\bar{\beta} = \frac{\gamma(\beta - \lambda\gamma)}{\beta - \lambda}, \qquad \bar{\gamma} = \frac{\beta^2(1 - \lambda)}{\beta - \lambda}.$$

Our goal is to choose β, γ, and λ so that $\bar{\beta} = \beta$, $\bar{\gamma} = \gamma$. If this is the case, the above derivation repeats for the next iteration, with y_{n-1} and y_n interchanged. It follows that in the transformed problem $\overline{\text{LP}}$, every step will be equivalent to the first, and in particular the decrease in the potential function, always using the same value for λ, will be the same on each iteration. Via straightforward manipulations we have

$$\beta = \bar{\beta} \Leftrightarrow \lambda = \frac{\beta(\beta - \gamma)}{\beta - \gamma^2} \tag{3.2}$$

$$\gamma = \bar{\gamma} \Leftrightarrow \lambda = \frac{\beta(\beta - \gamma)}{\beta^2 - \gamma}. \tag{3.3}$$

Note that λ given by (3.3) immediately satisfies $0 \leqq \lambda \leqq 1$. Since the values of λ in (3.2) and (3.3) must coincide, and the numerators in the two expressions are identical, we are led to the condition

$$\beta - \gamma^2 = \beta^2 - \gamma, \qquad \gamma(1-\gamma) = \beta(\beta-1). \tag{3.4}$$

Regarding $\beta > 1$ as a function of γ, $0 < \gamma < 1$, we obtain a unique value

$$\beta = \frac{1+\sqrt{1+4\gamma(1-\gamma)}}{2}. \tag{3.5}$$

Thus for any $0 < \gamma < 1$, if β is chosen via (3.5), and λ by (3.3) (giving the same value as (3.2)), we obtain $\bar{\beta} = \beta$, $\bar{\gamma} = \gamma$ following the step $y(\lambda)$, as desired.

We next examine the behavior of the potential function for the same step. We have

$$\begin{aligned} c^T e &= (n-3) + \gamma + \beta \\ c^T y(\lambda) &= (n-3)(1-\lambda/\beta) + \gamma(1-\lambda\gamma/\beta) + \beta(1-\lambda) \\ &= (n-3) + \gamma + \beta - \lambda[(n-3) + \gamma^2 + \beta^2]/\beta \\ &= [(n-3) + \gamma + \beta](1-\lambda/\beta), \end{aligned}$$

where the last equality uses the fact that γ and β are related by (3.4). Thus

$$\frac{c^T y(\lambda)}{c^T e} = 1 - \frac{\lambda}{\beta}. \tag{3.6}$$

Using (3.6), for $y(\lambda)$ as in (3.1), we obtain

$$F(\lambda) \equiv f(c, y(\lambda)) - f(c, e) = 3 \ln(1-\lambda/\beta) - \ln(1-\lambda\gamma/\beta) - \ln(1-\lambda). \tag{3.7}$$

One steplength strategy when implementing Karmarkar's algorithm is to select a fixed value of λ, typically greater than .9, and take a step using λ so long as the potential function decreases. If descent is not obtained for this initial value, λ is reduced by a "backtracking" procedure until descent is obtained. (As mentioned in § 2, implementations of Karmarkar's algorithm usually base the step on $g = \bar{c}_p$ rather than $g = \bar{c}_q$, but it still makes sense to think of a fixed λ for the step using $\bar{c}_q$.) We now show that for any $0 < \gamma < 1$, if β is chosen by (3.5), and λ by (3.3), then the potential function indeed decreases.

LEMMA 3.1. *Let $0 < \gamma < 1$, and let β be given by* (3.5). *Suppose that Karmarkar's algorithm is applied to* LP, *using λ from* (3.3) *on each iteration. Then for every $k \geqq 0$, the algorithm obtains $f(c, x^k) - f(c, x^0) = -k\delta$, where $\delta = \ln[(\beta^2-\gamma)/(\beta^2-\beta)] > 0$.*

Proof. Substituting (3.3) into (3.7), $F(\lambda)$ is given by

$$\begin{aligned} &3 \ln\left(1 - \frac{\beta-\gamma}{\beta^2-\gamma}\right) - \ln\left(1 - \frac{\gamma(\beta-\gamma)}{\beta^2-\gamma}\right) - \ln\left(1 - \frac{\beta(\beta-\gamma)}{\beta^2-\gamma}\right) \\ &\quad = -\ln(\beta^2-\gamma) + 3\ln[(\beta^2-\gamma) - (\beta-\gamma)] - \ln[(\beta^2-\gamma) - \gamma(\beta-\gamma)] \\ &\qquad - \ln[(\beta^2-\gamma) - \beta(\beta-\gamma)] \\ &\quad = -\ln(\beta^2-\gamma) + 3\ln[\beta(\beta-1)] - \ln[\beta(1-\gamma)] - \ln[\gamma(\beta-1)] \\ &\quad = \ln\left(\frac{\beta^3(\beta-1)^3}{(\beta^2-\gamma)\beta(\beta-1)\gamma(1-\gamma)}\right) \\ &\quad = \ln\left(\frac{\beta(\beta-1)}{\beta^2-\gamma}\right), \end{aligned}$$

where the second and final equalities both use $\gamma(1-\gamma)=\beta(\beta-1)$, from (3.4). The lemma follows from the fact that in the transformed problem the potential reduction is $\delta=-F(\lambda)$ on every step. □

For example, if $\gamma=.4$, then (3.5) gives $\beta=1.2$; (3.3) gives $\lambda=12/13\approx.9231$; and Lemma 3.1 gives $\delta\approx1.466$. Thus for $\gamma=.4$ and $\beta=1.2$, using the fixed value $\lambda=12/13$ (why not?) results in an identical transformed problem on each step, and gives a reduction in the potential function of roughly 1.5 on each step.

Of considerably more interest theoretically is the strategy of choosing λ according to a linesearch of $F(\lambda)$, $0\leqq\lambda\leqq1$. For $F(\cdot)$ as in (3.7), straightforward differentiation shows that the exact minimizer λ satisfies

$$\frac{3}{\beta-\lambda}=\frac{\gamma}{\beta-\lambda\gamma}+\frac{1}{1-\lambda}=\frac{\gamma(1-\lambda)+(\beta-\lambda\gamma)}{(\beta-\lambda\gamma)(1-\lambda)}.$$

Cross-multiplying, and collecting terms, results in a quadratic equation for λ:

$$\gamma\lambda^2-2(\gamma+\beta-\gamma\beta)\lambda+(3\beta-\gamma\beta-\beta^2)=0. \tag{3.8}$$

The solution of (3.8) can be presented in a more appealing form by first simplifying the discriminant term:

$$\begin{aligned}4(\gamma+\beta-\gamma\beta)^2-4\gamma(3\beta-\gamma\beta-\beta^2)&=4(\beta^2\gamma^2-\beta^2\gamma-\beta\gamma^2-\beta\gamma+\gamma^2+\beta^2)\\&=4(\gamma\beta(\gamma\beta-\gamma-\beta-1)+\gamma+\beta)\\&=4(\gamma\beta(\gamma\beta-1)+(\gamma+\beta)(1-\gamma\beta))\\&=4(1-\gamma\beta)(\gamma+\beta-\gamma\beta),\end{aligned}$$

where the second equality used the fact that γ and β are related by (3.4). The unique solution to (3.8) satisfying $0\leqq\lambda\leqq1$ is then given by

$$\lambda'=\frac{(\gamma+\beta-\gamma\beta)-\sqrt{(1-\gamma\beta)(\gamma+\beta-\gamma\beta)}}{\gamma}. \tag{3.9}$$

To summarize, at this point we have shown the following. For any choice of γ, $0<\gamma<1$, if β is chosen using (3.5), and λ by (3.3), then $\bar\beta=\beta$, $\bar\gamma=\gamma$, so in the transformed problem all steps are identical to the first (with y_{n-1} and y_n interchanged on every other iteration). On the other hand, for the same γ and β, the value λ' in (3.9) corresponds to exact minimization of the potential function. The natural question is whether γ may be chosen so that λ from (3.3) coincides with λ' from (3.9). In Fig. 1 we plot λ from (3.3), and λ' from (3.9), as functions of γ, where for each γ, β is chosen according to (3.5). The curves intersect at a unique γ^* in (0, 1), with $\gamma^*\approx.525$. Choosing $\gamma=\gamma^*$, and $\beta=\beta^*\approx1.207$ from (3.5), then causes exact minimization of the potential function to result in $\bar\gamma=\gamma$, $\bar\beta=\beta$, so that each step of the algorithm obtains the same reduction in the potential function (approximately 1.3) as the first step. As a consequence of this construction, we have proved the following theorem.

THEOREM 3.2. *For each $n\geqq3$ there is a problem* LP *such that Karmarkar's algorithm, applied to* LP *using real arithmetic and exact linesearch of the potential function, obtains $f(c,x^k)-f(c,x^0)=-k\delta^*$ for each $k\geqq0$, where $\delta^*\approx1.3$ is independent of n.*

Proof. Take $\gamma=\gamma^*\approx.525$ in Lemma 3.1. □

Richard Stone [6] has succeeded in obtaining analytic expressions for γ^* and β^*, as well as the resulting λ^* and δ^*. Letting $u=((2\sqrt3)-3)^{1/2}$, they are:

$$\begin{aligned}\gamma^*&=(\sqrt3-u)/2,\\\beta^*&=(\sqrt3+u)/2,\\\lambda^*&=(\sqrt3-1)(\sqrt3+u)/2,\\\delta^*&=\ln((\sqrt3+1)/(\sqrt3-1)).\end{aligned} \tag{3.10}$$

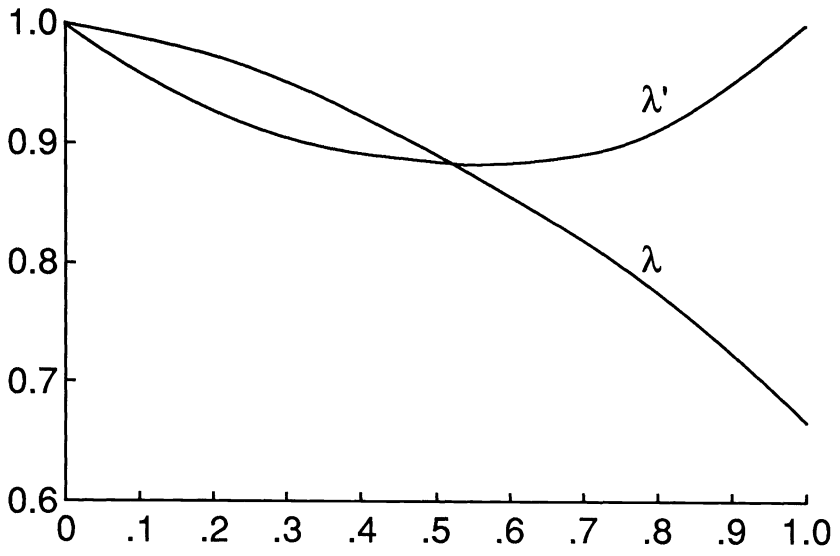


FIG. 1

Validity of these values can be established by direct substitution into (3.3), (3.4), (3.8), and the formula for δ in Lemma 3.1. We omit the details, but mention, for the interested reader, that verification is simplified by leaving "u" as a variable for as long as possible before substituting in the value given above.

Theorem 3.2 shows that any attempt to improve the worst-case complexity bound for Karmarkar's algorithm, with exact linesearch, cannot be based solely on showing that the reduction in the potential function somehow improves as the algorithm iterates. Ultimately, of course, what is important is not the potential function, but rather the objective value. We next turn to examine the objective values $c^T x^k$ for the class of problems analyzed in this section. For $0<\gamma<1$, let β be given by (3.5), and λ by (3.3), so that in the transformed problem the algorithm repeats itself on each iteration. By Lemma 3.1, there is then a constant $\delta>0$ so that the potential function is reduced by exactly δ on each iteration. To analyze the behavior of $c^T x^k$, we are led to obtain more information regarding the iterates x^k. The following lemma provides a complete characterization. For the remainder of the paper, we will find it convenient to define $\alpha=(1-\lambda/\beta)$. We will use $(\alpha)^k$ to denote powers of α, to avoid any possible confusion with superscripts, as in x^k.

LEMMA 3.3. *For $0<\gamma<1$, let β be given by* (3.5), *and suppose that the algorithm uses λ from* (3.3) *on each iteration. Let $\alpha=(1-\lambda/\beta)$. Then for each $k\geqq 1$, $x^k=(n/e^T\tilde{y}^k)\tilde{y}^k$ where $\tilde{y}^k\in\mathcal{R}^n$ is given by*

$$\tilde{y}_j^k=\begin{cases}1 & j=1\\ (\alpha)^k & 2\leqq j\leqq n\end{cases}$$

for k even, and

$$\tilde{y}_j^k=\begin{cases}1 & j=1\\ (\alpha)^k & 2\leqq j\leqq n-2\\ (1-\lambda\gamma/\beta)(\alpha)^{k-1} & j=n-1\\ (1-\lambda)(\alpha)^{k-1} & j=n\end{cases}$$

for k odd.

Proof. The proof is by induction on k. For $k=1$ the result is immediate, by (3.1). So assume that for some $k \geqq 1$ the result holds. On iteration k the step in the transformed problem is again of the form (3.1), except that y_{n-1} and y_n are interchanged if k is odd. Consequently let $\tilde{y}$ be the vector $y(\lambda)$, with the last two components interchanged if k is odd. Then by the definition of $T^{-1}(\cdot)$ in (2.1) we have

$$\begin{aligned} x^{k+1} &= n(e^\top X_k \tilde{y})^{-1} X_k \tilde{y} \\ &= n\left(\left(\frac{n}{e^\top \tilde{y}^k}\right) e^\top \tilde{Y}_k \tilde{y}\right)^{-1} \left(\frac{n}{e^\top \tilde{y}^k}\right) \tilde{Y}_k \tilde{y} \\ &= \left(\frac{n}{e^\top \tilde{Y}_k \tilde{y}}\right) \tilde{Y}_k \tilde{y}, \end{aligned}$$

where $\tilde{Y}_k$ is the diagonal matrix diag $(\tilde{y}^k)$. To complete the proof we need to show that $\tilde{y}_j^{k+1} = \tilde{y}_j^k \tilde{y}_j$, $j = 1, \cdots, n$. This is immediate for $j = 1, \cdots, n-2$, and also for $n-1 \leqq j \leqq n$ when k is even. However, for γ, β, and λ satisfying (3.3) and (3.4), it is straightforward to verify that $(1-\lambda)(1-\lambda\gamma/\beta) = (1-\lambda/\beta)^2 = (\alpha)^2$, which completes the argument for $n-1 \leqq j \leqq n$ when k is odd. □

LEMMA 3.4. *For $0 < \gamma < 1$, let β be given by (3.5), and suppose that the algorithm uses λ from (3.3) on each iteration. Let $\alpha = (1-\lambda/\beta)$. Then for each $k \geqq 1$,*

$$\frac{c^\top x^k}{c^\top x^0} = \frac{n(\alpha)^k}{1 + n(\alpha)^k - r^k},$$

where $r^k \to 0$ geometrically fast in k, independent of n.

Proof. We have $c^\top x^0 = (n-3) + \gamma + \beta$, and, from Lemma 3.4,

$$c^\top x^k = \frac{n(n-3+\gamma+\beta)(\alpha)^k}{1 + (n-1)(\alpha)^k}$$

for $k \geqq 2$, even, and

$$c^\top x^k = \frac{n((n-3)(\alpha)^k + [\gamma(1-\lambda\gamma/\beta) + \beta(1-\lambda)](\alpha)^{k-1})}{1 + (n-3)(\alpha)^k + [(1-\lambda\gamma/\beta) + (1-\lambda)](\alpha)^{k-1}}$$

for $k \geqq 1$, odd. However, for γ and β satisfying (3.4) it is straightforward to verify that

$$[\gamma(1-\lambda\gamma/\beta) + \beta(1-\lambda)] = (1-\lambda/\beta)(\gamma+\beta) = \alpha(\gamma+\beta).$$

Therefore, the lemma holds with

$$r^k = \begin{cases} (\alpha)^k & k \text{ even} \\ 3(\alpha)^k - [2 - \lambda(1+\gamma/\beta)](\alpha)^{k-1} & k \text{ odd.} \end{cases}$$ □

THEOREM 3.5. *For $n \geqq 3$, and $\varepsilon < 1$, there are linear programming problems such that Karmarkar's algorithm, applied using exact linesearch of the potential function, requires $k = \Theta(\ln(n/\varepsilon))$ iterations to obtain $c^\top x^k / c^\top x^0 \leqq \varepsilon$.*

Proof. Take $\gamma = \gamma^* \approx .525$, and let $\beta = \beta^* \approx 1.207$ be given by (3.5). Then the algorithm, using exact linesearch, repeats itself on each iteration, and Lemma 3.4 applies. Then $c^\top x^k / c^\top x^0 \leqq \varepsilon$ is equivalent to

$$\begin{aligned} n(\alpha)^k &\leqq \varepsilon(1 + n(\alpha)^k - r^k) \\ n(1/\varepsilon - 1)(\alpha)^k &\leqq 1 - r^k \\ k &\geqq \mu[\ln(1/\varepsilon - 1) + \ln(n) - \ln(1 - r^k)], \end{aligned}$$

where $\mu = -1/\ln(\alpha) > 0$. □

Theorem 3.5 proves that in the worst case, Karmarkar's algorithm, using exact linesearch, requires at least $\Omega(\ln(n/\varepsilon))$ iterations to reduce the gap to a factor ε of its initial value. Note that the role of $\ln(n)$ in this bound is similar to set-up charge, or fixed cost; in particular, we have *not* demonstrated a lower bound of $\Omega(\ln(n)\ln(1/\varepsilon))$ iterations. For the problems considered here, the bound of $\Theta(\ln(n/\varepsilon))$ iterations, in Theorem 3.5, should be compared to the bound of $O(n\ln(1/\varepsilon))$ iterations implied by the constant potential reduction shown for the same problems in Theorem 3.2. These bounds, taken together, demonstrate that potential reduction may provide a poor indication of the algorithm's actual convergence. Analogous results hold for an implementation of the algorithm using a "fixed fraction to the boundary" step rule, fixing $\lambda \in (\frac{2}{3}, 1)$. The only change required is the choice of γ—note that by Fig. 1 it is clear that for any $\lambda \in (\frac{2}{3}, 1)$ there is a unique $0 < \gamma < 1$, and β from (3.5), so that the given λ agrees with λ from (3.3). Lemma 3.1 then applies, as does a straightforward analog of Theorem 3.5.

The difference between the constant potential reduction of Lemma 3.1, and the actual objective performance of Lemma 3.4, can be even more striking when the step in the transformed problem is viewed in terms of $\bar{c}_p$ rather than $\bar{c}_q$ (see § 2 for the distinction). For example, consider the case of $\gamma = \beta = 1$, as used in [5]. By the analysis at the beginning of this section, it is clear that the algorithm repeats itself for any $0 < \lambda < 1$; in fact, $\lambda = 1$ leads to the optimal solution in one step. Take $\lambda = \frac{1}{2}$, so on each step $y(\lambda) = (1, .5, \cdots, .5)^T$, and $e^T y(\lambda) = 1 + (n-1)/2$. Rescaling to remain on the simplex S, in the transformed problem, then obtains the point $\hat{y} = [n/(n+1)](2, 1, \cdots, 1)^T$. Note that $\|\hat{y} - e\| = R/(n+1) < r$, where $r = \sqrt{n/(n-1)}$ and $R = \sqrt{n(n-1)}$ denote the radii of spheres, centered at e, which inscribe and circumscribe S. The move from e to $\hat{y}$ corresponds to a step in the direction $-c_p$ of norm less than r, compared to the step of length R, which would lead immediately to the solution. Thus a "fraction to the boundary" of $\lambda = \frac{1}{2}$, using $\bar{c}_q$, corresponds to a "fraction to the boundary" of only $1/(n+1)$, using $\bar{c}_p$. Viewed in terms of $\bar{c}_p$, this is an extremely short step. However, Lemmas 3.3 and 3.4 continue to apply, with $\lambda = \frac{1}{2}$, so the algorithm still requires only $\Theta(\ln(n/\varepsilon))$ steps to reduce the gap to a fraction ε of its initial value.

Acknowledgment. The author would like to thank Richard Stone, for obtaining the closed-form solutions in (3.10); Jeff Lagarias and an anonymous referee, for suggestions which considerably improved the presentation of Lemmas 3.3 and 3.4, and Theorem 3.5; and Mike Todd, for a conversation which led to the observation regarding the step viewed in terms of $\bar{c}_p$, at the end of § 3.

REFERENCES

[1] K. M. ANSTREICHER (1989), *The worst-case step in Karmarkar's algorithm*, Math. Oper. Res., 14, pp. 294–302.

[2] M. D. ASIC, V. V. KOVACEVIC-VUJCIC, AND M. D. RADOSAVLJEVIC-NIKOLIC (1990), *Asymptotic behavior of Karmarkar's method for linear programming*, Math. Programming, 46, pp. 173–190.

[3] D. BAYER AND J. C. LAGARIAS (1987), *Karmarkar's algorithm and Newton's method*, AT & T Bell Laboratories, Murray Hill, NJ; Math. Programming, to appear.

[4] N. KARMARKAR (1984), *A new polynomial time algorithm for linear programming*, Combinatorica, 4, pp. 373–395.

[5] C. MCDIARMID (1990), *On the improvement per iteration in Karmarkar's algorithm for linear programming*, Math. Programming, 46, pp. 299–320.

[6] R. STONE (1990), Private communication.

[7] M. J. TODD (1989), Private communication.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 30-41, February 1991

004

COMPOSITE NONSMOOTH PROGRAMMING WITH GÂTEAUX DIFFERENTIABILITY*

V. JEYAKUMAR†

Abstract. This paper examines constrained nonsmooth optimization problems where the objective function and the constraints are compositions of locally Lipschitz functions and Gâteaux differentiable functions, but are not necessarily Fréchet differentiable or strict differentiable. Lagrangian necessary and sufficient optimality conditions are presented for various classes of composite programs. These are obtained by constructing appropriate convex approximations for composite functions. The Lagrange multipliers are also characterized in terms of subgradients of the value function under appropriate conditions.

Key words. nonsmooth optimization, optimality conditions, convex approximations, Clarke subdifferentials, convex composite problems

AMS(MOS) subject classifications. 90C30, 49B27, 49A52, 90C48

1. Introduction. In this paper we study composite nondifferentiable programming problems of the form

$$\text{(P)} \qquad \begin{aligned} &\text{minimize} && f_0(F_0(x)) \\ &\text{subject to} && x \in X, \quad f_i(F_i(x)) \leqq 0, \quad i = 1, 2, \cdots, m, \end{aligned}$$

where X is a real Banach space, $f_0, \cdots, f_m$ are real-valued locally Lipschitz functions on R^n, and for each $i = 0, 1, \cdots, m$, $F_i : X \to R^n$ is locally Lipschitz and Gâteaux differentiable with Gâteaux derivative $F'(\cdot)$, but not necessarily continuously Fréchet differentiable or strict differentiable [5]. Many applications of optimization concern objective functions and constraints that are not necessarily smooth, but are of "composite" type. The model problem (P) covers a very wide range of practical optimization problems, which often arise in penalty methods or restricted step methods for constrained optimization, and in minimax problems (see [7], [11], [4], [3]). Convex composite problems (P) were examined in regard to Lagrangian conditions in [7] and [11] for the case in which the functions F_i, $i = 0, 1, \cdots, m$, are assumed to be continuously Fréchet differentiable.

Studies of optimization problems have led in recent years to the development of a nonsmooth calculus (e.g., [5], [16], [20]-[21], [23]-[24]). In particular, the Clarke subdifferential calculus has proved to be a potent and flexible tool in mathematical programming. However, the lack of generalized calculus such as chain rules (cf. [5, Thm. 2.3.10]) for Gâteaux differentiable functions is one of the chief reasons why the study of composite programming problems has so far been limited mainly to problems (P) in which the functions F_i, $i = 0, 1, \cdots, m$ are assumed to be *continuously Fréchet* differentiable or *strict* differentiable. In this paper, it is shown how Clarke subdifferentials can be used to study the composite model problems (P) with locally Lipschitz and Gâteaux differentiable functions.

This paper examines various classes of composite nondifferentiable programming problems. First, the composite model problem (P) is analysed in regard to Lagrangian necessary optimality conditions and Gâteaux differentiability assumptions. The

* Received by the editors November 20, 1989; accepted for publication (in revised form) August 1, 1990. This research was partially supported by Australian Research Council grant A68930162.

† Department of Applied Mathematics, University of New South Wales, Kensington, New South Wales, Australia.

necessary conditions are obtained in an easily verifiable form by first establishing appropriate upper convex approximations ([11], [13]; see also the definition below) for the composite functions. This approach allows us to relax the usual continuously Fréchet differentiability or strict differentiability assumptions used in the literature [7], and leads to a generalized chain rule for locally Lipschitz and Gâteaux differentiable functions. Second, it is shown that the necessary conditions become sufficient for *global* optimality under appropriate hypotheses in the case where for each $i = 1, \cdots, m$, f_i is convex, and $F_i = F$. A duality theorem is also given for this class of (nonconvex) composite programs. Finally, a complete characterization of the Lagrange multipliers is given in terms of subgradients of the value function for (P) in which, for each $i = 0, 1, 2, \cdots, m$, f_i is convex and $F_i = F$, and the range of F is assumed to be convex. These problems provide a class of programs which are not convex but possess many of the nice properties that convex programs have.

2. Gâteaux derivatives, composite functions, and convex approximations. In this section we present various properties of generalized derivatives of locally Lipschitz functions and new versions of upper convex approximations for the composite function g of the form $g := f \circ F$, where f is a real-valued locally Lipschitz function on R^n, and $F: X \to R^n$ is a locally Lipschitz and Gâteaux differentiable function on a real Banach space X. These properties deal with generalized subdifferentials for locally Lipschitz functions due to Clarke [5], and Michel and Penot [16].

For a real locally Lipschitz function $h: X \to R$, the Clarke directional derivative [5], the Michel-Penot directional derivative [16], and the upper Dini-directional derivative are, respectively, defined by

$$h^0(a, x) := \lim_{d \to 0} \sup_{\lambda \downarrow 0} \lambda^{-1}[h(a + d + \lambda x) - h(a + d)],$$

$$h^\diamond(a, x) := \sup_{z \in X} \limsup_{\lambda \downarrow 0} \lambda^{-1}[h(a + \lambda z + \lambda x) - h(a + \lambda z)],$$

$$h^+(a, x) := \limsup_{\lambda \downarrow 0} \lambda^{-1}[h(a + \lambda x) - h(a)].$$

The Clarke subdifferential and Michel-Penot subdifferential are, respectively, defined by

$$\partial^0 h(a) := \{v \in X' \mid h^0(a, x) \geqq v(x), \forall x \in X\},$$

and

$$\partial^\diamond h(a) := \{v \in X' \mid h^\diamond(a; x) \geqq v(x), \forall x \in X\}.$$

Then, $h^0(a, \cdot)$, and $h^\diamond(a, \cdot)$ are continuous sublinear functions with $h^+(a, x) \leqq h^\diamond(a, x) \leqq h^0(a, x)$, $x \in X$, and $h^0(a, x) = \limsup_{y \to a} h^\diamond(y, x)$. The subdifferentials $\partial^0 h(a)$ and $\partial^\diamond h(a)$ are nonempty convex weak* compact subsets of X', the topological dual space of X. These subdifferentials satisfy the relation that $\partial^\diamond h(a) \subset \partial^0 h(a)$. When h is convex, $\partial^\diamond h(a) = \partial^0 h(a) = \partial h(a)$, the subdifferential in the sense of convex analysis. The Clarke subdifferential mapping $x \to \partial^0 h(x)$ is weak* upper semicontinuous; whereas the Michel-Penot subdifferential has the property that h is Gâteaux differentiable at a if and only if $\partial^\diamond h(a) = \{h'(a)\}$. The locally Lipschitz function $h: X \to R$ is said to be *regular* [5] at $a \in X$ if the usual directional derivative $h'(a, \cdot)$ at a exists and $h'(a, x) = h^0(a, x)$, for all $x \in X$. Following Ioffe [11] (see also Jeyakumar [13]), we say that a function $\psi_a: X \to R$, $a \in X$, is an *upper convex approximation* for h at a if ψ_a is a continuous sublinear function and $h^+(a, x) \leqq \psi_a(x)$, for each $x \in X$. It is clear that $h^\diamond(a, \cdot)$, and $h^0(a, \cdot)$ are upper convex approximations for h at a. We also note that

the Clarke directional derivative has the useful property that for each $x \in X$, the mapping $y \to h^0(y, x)$ is *upper semicontinuous at a* and serves as a (semi-)continuous upper convex approximation for h. As we shall see later, this property turns out to be helpful in applications. The Michel-Penot directional derivative mapping $y \to h^\diamond(y, x)$ is not, in general, upper semicontinuous. The next proposition presents a characterization result for upper semicontinuity of $h^\diamond(\cdot, x)$ at a point.

PROPOSITION 2.1. *Let $a \in X$, and let $h: X \to R$ be locally Lipschitz at a. Then, for each $x \in X$, $h^\diamond(\cdot, x)$ is upper semicontinuous at $a \in X$ if and only if $h^0(a, \cdot) = h^\diamond(a, \cdot)$.*

Proof. Let $x \in X$. If $h^\diamond(\cdot, x)$ is upper semicontinuous at a then $\limsup_{y \to a} h^\diamond(y, x) \leqq h^\diamond(a, x)$, and so, $\limsup_{y \to a} h^\diamond(y, x) \leqq h^\diamond(a, x) \leqq h^0(a, x)$. The equality $h^\diamond(a, x) = h^0(a, x)$, will now follow from the property that $h^0(a, x) = \limsup_{y \to a} h^\diamond(y, x)$. Conversely, if the equality holds for each $x \in X$, then

$$\limsup_{y \to a} h^\diamond(y, x) \leqq h^0(a, x) \leqq h^\diamond(a, x),$$

which establishes the upper semicontinuity of $h^\diamond(\cdot, x)$ at a. □

It should be noted that if the function h is regular at a then it satisfies the equality that $h^0(a, x) = h^\diamond(a, x)$, for each $x \in X$. Moreover, if ψ_a is allowed to be an extended real-valued function then the *radial* generalized directional derivative [10], [20] of h at a can also be chosen as an upper convex approximation for h at a for continuous functions h. For the properties of this directional derivative and its associated generalized subdifferential, see [10], [20], and [22].

We now see that for the locally Lipschitz function h, the sublinear function defined by $\psi_a(x) := p^\diamond(a, x) - q'(a)x$ is a better upper convex approximation of h at a than the above-mentioned approximations, where $q: X \to R$ is locally Lipschitz and Gâteaux differentiable at a, and $p = h + q$. This follows from the fact that $p^\diamond(a, \cdot)$ is an upper convex approximation of p and a, and the property that $p^\diamond(a, x) \leqq h^\diamond(a, x) + q^\diamond(a, x) = h^\diamond(a, x) + q'(a)x$. This observation leads us to construct the following more general class of upper convex approximations for locally Lipschitz functions. We first note that the function h is said to be *quasi-differentiable* at a point a in the sense of Pshenichnyi [17] if h is directionally differentiable and $h'(a, x) = \max_{v \in \partial^* h(a)} v(x)$, for some convex weak* compact set $\partial^* h(a)$.

PROPOSITION 2.2. *Let $q: X \to R$ be locally Lipschitz and quasi-differentiable at a with the convex weak* compact set $\partial^* q(a)$. Let $p = h + q$. Then for each $v \in \partial^* q(a)$, the sublinear function defined by*

$$\psi_a(x) := p^\diamond(a, x) - v(x)$$

is an upper convex approximation of h at a.

Proof. Since q is directionally differentiable,

$$h^+(a, x) \leqq p^+(a, x) - q'(a, x) \leqq p^+(a, x) - v(x),$$

for each $v \in \partial^* q(a)$. The conclusion now follows by observing that $p^+(a, x) \leqq p^\diamond(a, x)$. □

It is worth noting that the latter classes of approximations may be used to establish tighter optimality conditions for optimization problems. Recall that if a function h attains its minimum at a and admits an upper convex approximation $\psi_a(\cdot)$ at a, then $0 \in \partial\psi_a(0)$. We now present a general chain rule for differentiation with Gâteaux differentiability. Let us first note that if $f: R^n \to R$ and $F: X \to R^n$ are locally Lipschitz, then the composite function $g := f \circ F$ is locally Lipschitz. The open unit ball in R^n is denoted by B.

THEOREM 2.1 (generalized chain rule). *Let $F: X \to R^n$ be locally Lipschitz near $x \in X$ and Gâteaux differentiable at x; let $f: R^n \to R$ be locally Lipschitz near $F(x)$. Assume that for each y, ψ_y is an upper convex approximation for f at y. If, for each $d \in X$, $y \to \psi_y(d)$ is upper semicontinuous at $F(x)$ then*

$$\partial^{\diamond}(f \circ F)(x) \subset \{w^T F'(x) \mid w \in \partial\psi_{F(x)}(0)\}. \tag{2.1}$$

Moreover, if f is regular at $F(x)$ and if ψ is chosen as the Clarke directional derivative then equality holds in (2.1) *with the subdifferential of $\psi_{F(x)}$ at zero replaced by the Clarke subdifferential of f at $F(x)$.*

Proof. Let $A = \{w^T F'(x) \mid w \in \partial\psi_{F(x)}(0)\}$. Then, the support function of A, evaluated at a point h in X, is given by

$$\pi_x(h) := \max\left\{ \sum_{i=1}^{n} w_i F_i'(x)h \;\middle|\; w \in \partial\psi_{F(x)}(0) \right\}. \tag{2.2}$$

The conclusion will follow if we show that for any $h \in X$,

$$(f \circ F)^{\diamond}(x, h) \leqq \pi_x(h).$$

From the mean-value theorem of Studniarski [22, Thm. 4.3], for any h, k in X and t small positive,

$$\begin{aligned} t^{-1}[f \circ F(x+th+tk) - (f \circ F)(x+tk)] &= t^{-1}[f(F(x+th+tk)) - f(F(x+tk))] \\ &= t^{-1} v_z^T (F(x+th+tk) - F(x+tk)), \end{aligned}$$

for some $v_z \in \partial\psi_z(0)$, $z \in [F(x+tk),\ F(x+tk+th)]$. Now, fix $h, k \in X$; let $\varepsilon > 0$. Then,

$$\begin{aligned} &|t^{-1}[F_i(x+th+tk) - F_i(x+tk)] - F_i'(x)h| \\ &\quad = |t^{-1}[F_i(x+th+tk) - F_i(x)] - F_i'(x)(h+k) \\ &\qquad + t^{-1}[F_i(x) - F_i(x+tk)] + F_i'(x)k| \\ &\quad \leqq |t^{-1}[F_i(x+th+tk) - F_i(x)] - F_i'(x)(h+k)| \\ &\qquad + |t^{-1}[F_i(x) - F_i(x+tk)] + F_i'(x)k| \\ &\quad = |t^{-1}[F_i(x+th+tk) - F_i(x)] - F_i'(x)(h+k)| \\ &\qquad + |t^{-1}[F_i(x+tk) - F_i(x)] - F_i'(x)k|. \end{aligned}$$

Thus,

$$|t^{-1}[F_i(x+th+tk) - F_i(x+tk)] - F_i'(x)h| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \tag{2.3}$$

for sufficiently small $t > 0$. Since $f: R^n \to R$ and for each $d \in Y$, $y \to \psi_y(d)$ is upper semicontinuous at $F(x)$, it follows from the same arguments as in Proposition 2.1.5 of [5] that, for sufficiently small $t > 0$,

$$\partial\psi_z(0) \subset \partial\psi_{F(x)}(0) + \varepsilon B,$$

and so,

$$v_z \in \partial\psi_{F(x)}(0) + \varepsilon B.$$

Therefore, for any t sufficiently small positive,

$$\begin{aligned} t^{-1}[f \circ F(x+th+tk) - f \circ F(x+tk)] &= \sum_{i=1}^{n} v_z^i t^{-1}[F_i(x+th+tk) - F_i(x+tk)] \\ &\leqq \pi_x(h) + \left(nK_0 + \sum_{i=1}^{n} K_i \|h\| + \varepsilon \right) \varepsilon, \end{aligned}$$

where K_0 is the Lipschitz constant of $\psi_{F(x)}$ near zero, and K_i, $i=1,2,\cdots,n$ are the Lipschitz constants of F_i's near x. Hence,

$$(f\circ F)^{\diamond}(x,h)\leqq \pi_x(h),$$

and so the inclusion (2.1) holds.

If f is further assumed to be regular at $F(x)$ then the equality in (2.1) is obtained by adapting the same lines of arguments as in Theorem 2.3.9 of Clarke [5]. The details are left to the reader. □

An inclusion of the form (2.1) is obtained in [1] under the assumption that $\partial^{\diamond}f(F(x))=\partial^0 f(F(x))$. The functions f which satisfy this equality are called "*normal*" at $F(x)$ in [1]. A related chain rule is proved in [16] under a stronger differentiability assumption. The methods of proof used in our Theorem 2.1 and Theorem 2.3.9 of Clarke [5] suggest a more general chain rule for locally Lipschitz functions without any differentiability assumptions. This will be presented in [14].

We now see that the generalized chain rule leads us to a general upper convex approximation for the composite function $g:=f\circ F$ using the Clarke subdifferential. A simple chain rule for convex composite functions (see [1]) also easily follows from Theorem 2.1.

COROLLARY 2.1 (upper convex approximation with Gâteaux differentiability). *Let $F:X\to R^n$ be locally Lipschitz near $x\in X$ and Gâteaux differentiable at x; let $f:R^n\to R$ be locally Lipschitz near $F(x)$. Then, the function π_x defined by*

$$\pi_x(h):=\max\left\{\sum_{i=1}^{n} w_iF_i'(x)h \;\middle|\; w\in\partial^0 f(F(x))\right\} \tag{2.4}$$

is an upper convex approximation of $f\circ F$ at x.

Proof. Define $\psi_y(h):=f^0(y,h)$. Then, for each y, ψ_y is an upper convex approximation for f at y, and for each $d\in X$, $y\to\psi_y(d)$ is upper semicontinuous at $F(x)$. Then, the conclusion follows from Theorem 2.1 by noting that $\partial\psi_{F(x)}(0)=\partial^0 f(F(x))$. □

COROLLARY 2.2. *Let $F:X\to R^n$ be locally Lipschitz and Gâteaux differentiable at $x\in X$; let $f:R^n\to R$ be a convex function. Then,*

$$\partial^{\diamond}(f\circ F)(x)=\{v^TF'(x)\,|\,v\in\partial f(F(x))\}$$

and the directional derivative of $f\circ F$ exists at x.

Proof. The conclusion follows from Theorem 2.1 by noting that the convex function on R^n is regular at $F(x)$. □

3. Necessary Lagrangian conditions. Consider now the constrained composite minimization problem with a convex set constraint

$$\text{(PC)}\qquad \begin{aligned} &\text{minimize} && f_0(F_0(x))\\ &\text{subject to} && x\in C,\quad f_i(F_i(x))\leqq 0,\quad i=1,2,\cdots,m, \end{aligned}$$

where C is a convex subset of X, f_i, and F_i, $i=0,1,\cdots,m$ are as in problem (P). The functions F_i, $i=0,1,\cdots,m$ are *not* assumed to be continuously Fréchet differentiable (or strictly differentiable). For (PC) with $C=X=R^n$, but functions F_i, $i=0,1,\cdots,m$, continuously differentiable, necessary Lagrangian conditions are known (see [7]). In this section we show how necessary Lagrangian conditions for the general problem (PC) can directly be obtained with Clarke subdifferentials using upper convex approximations.

In passing, we note that the composite function h of the form $h(x):=g(x)+f\circ F(x)$, where $g:X\to R^n$, and $F:X\to R^n$ are Gâteaux differentiable and

$f: R^n \to R$ is convex, can be expressed simply in the form of $\tilde{f} \circ \tilde{F}(x)$ for some suitable locally Lipschitz function $\tilde{f}$ and Gâteaux differentiable function $\tilde{F}$.

The following *alternative theorem* will be required. It is an immediate special case of an alternative theorem in [6] and [12].

LEMMA 3.1. *Let $\triangle \subset X$ be convex, and let $g_1, \cdots, g_p$ be real-valued continuous convex functions on X. Then exactly one of the following systems is consistent*:

$$\text{(3.1)} \qquad \begin{aligned} &\text{(i)} \quad \exists x \in \triangle, \quad g_1(x) < 0, \cdots, g_p(x) < 0, \\ &\text{(ii)} \quad (\exists \lambda \in R^p, \lambda \geqq 0, \lambda \neq 0) \quad (\forall x \in \triangle) \quad \sum_{i=1}^{p} \lambda_i g_i(x) \geqq 0. \end{aligned}$$

For a convex set C and $a \in C$, we denote by cone $(C-a) := \{\alpha(c-a) \mid c \in C, \alpha \geqq 0\}$, the cone generated by C at a, and $C^+ = \{v \in X' \mid v(x) \geqq 0, \forall x \in C\}$, the dual cone of C. The following Fritz John type theorem holds for (PC).

THEOREM 3.1. *Let a be feasible for* (PC), *and let $\psi^i_{F_i(a)}$ be an upper convex approximation for f_i at $F_i(a)$. Suppose that the function $y \to \psi^i_y$ is upper semicontinuous at $F_i(a)$. If the problem* (PC) *attains a local minimum at $a \in C$, then there exist Lagrange multipliers $\tau \geqq 0$, $\lambda_i \geqq 0$, $i = 1, 2, \cdots, m$, not all zero, and $v_i \in \partial \psi^i_{F_i(a)}(0)$, $i = 0, 1, 2, \cdots, m$, satisfying*

$$\text{(3.2)} \qquad \begin{aligned} &\left(\tau v_0^T F_0'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F_i'(a)\right)(x-a) \geqq 0 \quad \forall x \in C, \\ &\lambda_i f_i(F_i(a)) = 0, \qquad i = 1, 2, \cdots, m. \end{aligned}$$

Proof. Let $I(a) = \{i \mid f_i(F_i(a)) = 0, i = 1, 2, \cdots, m\}$ and let $g_i = f_i \circ F_i$. Suppose that the following system has a solution:

$$\text{(3.3)} \qquad d \in X, \quad d \in \text{cone}\,(C-a), \quad \pi^i_a(d) < 0, \quad i \in I(a) \cup \{0\},$$

where π^i_a is defined by (see (2.2))

$$\pi^i_a(d) := \max\left\{ \sum_{i=1}^{n} w_i F_i'(a) d \;\middle|\; w \in \partial \psi^i_{F_i(a)}(0) \right\}.$$

Then, the system

$$\text{(3.4)} \qquad d \in X, \quad d \in \text{cone}\,(C-a), \quad g_i^+(a, d) < 0, \quad i \in I(a) \cup \{0\}$$

has a solution $d \in X$. Hence, there exists $\alpha_0 > 0$ such that

$$a + \alpha d \in C, \quad g_0(a+\alpha d) < g_0(a), \quad g_i(a+\alpha d) < g_i(a) = 0,$$

whenever $0 < \alpha \leqq \alpha_0$, and $i \in I(a)$. Since $g_i(a) < 0$, for $i \notin I(a)$ and g_i is continuous in a neighbourhood of a, there exists $\alpha_1 > 0$ such that $g_i(a+\alpha d) < 0$, whenever $0 < \alpha \leqq \alpha_1$, and $i \notin I(a)$. Now, let $\bar{\alpha} = \min\{\alpha_0, \alpha_1\}$. Then, $a + \alpha d$ is a feasible solution for (PC) and $g_0(a+\alpha d) < g_0(a)$, for sufficiently small α such that $0 < \alpha \leqq \bar{\alpha}$. This contradicts the **local minimum** of (PC) at $x = a$. Hence, the system (3.3) has no solution. Since, for each i, $\pi^i_a(\cdot)$ is a sublinear function and cone $(C-a)$ is convex, it follows from Lemma 3.1 that there exist $\tau \geqq 0$ and $\lambda_i \geqq 0$, $i = 1, 2, \cdots, m$, not all zero, such that

$$\tau \pi^0_a(x) + \sum_{i \in I(a)} \lambda_i \pi^i_a(x) \geqq 0 \quad \forall x \in \text{cone}\,(C-a).$$

Then, by a separation theorem and by choosing $\lambda_i = 0$, for $i \notin I(a)$, we get

$$0 \in \tau \partial \pi^0_a(0) + \sum_{i=1}^{m} \lambda_i \partial \pi^i_a(0) - (C-a)^+.$$

Hence, there exist $v_i \in \partial\psi^i_{F_i(a)}(0)$, $i = 0, 1, \cdots, m$ such that

$$\tau v_0^T F_0'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F_i'(a) \in (C-a)^+.$$

The conclusion now follows from the definition of the dual cone and λ_i, for $i \notin I(a)$. □

Necessary conditions of Kuhn-Tucker type follow from Theorem 3.1, under any constraint qualification that ensures $\tau \neq 0$. In particular, the following *Slater* type constraint qualification does this:

$$\exists x_0 \in \text{cone}\,(C-a), \quad v^T F_i'(a) x_0 < 0 \quad \forall v \in \partial\psi^i_{F_i(a)}(0), \quad i \in I(a),$$

where $I(a) = \{i \mid f_i(F_i(a)) = 0\}$; thus, if the problem (PC) attains a local minimum at $a \in C$ and a suitable constraint qualification holds at a, then there exist Lagrange multipliers $\lambda_i \geqq 0$, $i = 1, 2, \cdots, m$, and $v_i \in \partial\psi_{F_i(a)}(0)$, $i = 0, 1, 2, \cdots, m$, satisfying

$$\left(v_0^T F_0'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F_i'(a)\right)(x-a) \geqq 0 \quad \forall x \in C$$

and $\lambda_i f_i(F_i(a)) = 0$, $i = 1, 2, \cdots, m$.

We now show how Clarke subdifferentials can be used to describe necessary conditions for the composite model problem (P) with locally Lipschitz and Gâteaux differentiable functions. The following Kuhn-Tucker type optimality conditions with Clarke subdifferentials for (P) easily follow from Theorem 3.1 by taking $C = X$ and choosing the Clarke directional derivatives as upper convex approximations.

COROLLARY 3.1. *Assume that the problem* (P) *attains its local minimum at* $x = a \in X$. *If there exists* $x_0 \in X$ *such that* $v^T F_i'(a) x_0 < 0$, *for all* $v \in \partial^0 f_i(F_i(a))$, $i \in I(a)$, *then there exist Lagrange multipliers* $\lambda_i \geqq 0$, $i = 1, 2, \cdots, m$, *and* $v_i \in \partial^0 f_i(F_i(a))$, $i = 1, 2, \cdots, m$, *satisfying*

$$v_0^T F_0'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F_i'(a) = 0, \qquad \lambda_i f_i(F_i(a)) = 0, \qquad i = 1, 2, \cdots, m. \tag{3.5}$$

It is worth noting that the Clarke calculus [5] cannot directly be applied to our composite model problem (P). We have seen that our upper convex approximation scheme and our analysis in § 2 pave the way for use of Clarke subdifferentials for general composite nonlinear programming problems. Moreover, we shall show in the next section that the condition (3.5) becomes sufficient for global optimum for certain class of nonconvex problems.

4. Convex composite programs. In this section, we examine sufficient conditions for global (and local) optimum of convex composite programs. We first consider

$$\text{(UP)} \qquad \begin{aligned} &\text{minimize} && f(F(x)) \\ &\text{subject to} && x \in C \end{aligned}$$

where $f: R^n \to R$ is a convex function, $F: X \to R^n$ is a locally Lipschitz and Gâteaux differentiable function on X, and C is a convex subset of X. Using a mean value theorem [2] and Theorem 2.1, we shall present a sufficient condition for a *local* minimum of (UP).

PROPOSITION 4.1. *For the problem* (UP), *let* $a \in C$. *If there exists a neighbourhood* $N(a)$ *of* a *such that*

$$(\forall x \in C \cap N(a) \backslash \{a\}) \quad (\forall v \in \partial f(F(x))) \quad v^T F'(x)(x-a) > 0, \tag{4.1}$$

then a *is a strict local minimum for* (UP).

Proof. Suppose that a is not a strict local minimum for (UP). Then, there exists $y \in N(a) \cap C$ such that $y \neq a$, and $f(F(y)) \leqq f(F(a))$. From the mean value theorem [2], there exists $0 < \alpha < 1$ such that $z = y + \alpha(a-y) \in C \cap N(a)$, $z \neq a$, and $0 \leqq f \circ F(a) - f \circ F(y) = w^T(a-y)$, for some $w \in \partial^{\diamond}(f \circ F)(z)$. Now, by Theorem 2.1 (see also Corollary 2.2), there exists $v \in \partial f(F(z))$ such that $w^T(a-y) = v^T F'(z)(a-y) \geqq 0$. Since $a - z = (1-\alpha)(a-y)$, $v^T F'(z)(a-z) \geqq 0$. This contradicts our assumption (4.1). □

Remark 4.1. In proving Proposition 4.1, we only used the property that f is locally Lipschitz on R^n. The result therefore holds for locally Lipschitz function f by replacing $\partial f(F(x))$ by $\partial^0 f(F(x))$ or $\partial \psi_{F(x)}(0)$ in (4.1). For related sufficient conditions for locally Lipschitz functions, see Chaney [4].

Let us now examine conditions for *global* minimum for (UP).

PROPOSITION 4.2 (conditions for global minimum). *Consider the problem* (UP). *Let* $a \in C$. *If* $F'(a)(\operatorname{clcone}(C-a)) = R^n$, *then the following statements are equivalent*:

$$\text{(4.2)} \qquad \text{(i)} \quad f(F(a)) = \min_{x \in C} f(F(x)),$$

$$\text{(4.3)} \qquad \text{(ii)} \quad (\exists v \in \partial f(F(a))) \quad (\forall x \in C) \quad v^T F'(a)(x-a) \geqq 0.$$

Proof. ((4.2)⇒(4.3)). This follows from the results of the previous section.

((4.3)⇒(4.2)). Let $x \in C$; let $F(x) = y$ and $F(a) = \bar{y}$. From the convexity of f, we get

$$\begin{aligned} f(F(x)) - f(F(a)) &= f(y) - f(\bar{y}) \\ &\geqq v^T(y - \bar{y}). \end{aligned}$$

By our assumption, $F'(a)z = y - \bar{y}$ is solvable for $z \in \operatorname{clcone}(C-a)$. Hence, $v^T F'(a)(\operatorname{clcone}(C-a)) \subset R_+$, and so,

$$f(F(x)) - f(F(a)) \geqq v^T F'(a) z \geqq 0.$$

Thus, (4.2) holds. □

Remark 4.2. When $C = X$, our hypothesis in Proposition 4.2 requires that the Gâteaux derivative of F at a is *onto*. However, it provides new and easily verifiable conditions for global minima of a class of nonconvex problems. We now give an extremely simple example, merely to provide some intuitive grasp for the kind of problem that falls within the scope of Proposition 4.2. Consider $f(x, y) = 3x^2 - 2xy + 4y^2$, and $F(x, y) = (x - x^3, y + y^3)$. Then the hypothesis of Proposition 4.2 is satisfied with $C = R^2$, e.g., at $a = (1, -1)$. Proposition 4.2 extends a well-known characterization of optimality result for convex programs to composite (nonconvex) problems.

Now, consider the following *special class of convex* composite programs:

$$\text{(PF)} \qquad \begin{aligned} &\text{minimize} && f_0(F(x)) \\ &\text{subject to} && x \in X, \quad f_i(F(x)) \leqq 0, \quad i = 1, 2, \cdots, m, \end{aligned}$$

where $F: X \to R^n$ is a locally Lipschitz and Gâteaux differentiable function, and $f_i: R^n \to R$, $i = 0, 1, 2, \cdots, m$ are convex functions. We shall see that for this class of programs the necessary condition (3.5) becomes sufficient for global minimum under an additional assumption on the derivative of F at the optimum.

The *null space* of a function f is denoted by $N[f]$. Note that, by the Farkas theorem (see [6, Thm. 2.2.6]), for each fixed x, $a \in X$, $F(x) - F(a) \in F'(a)(X)$ if and only if $F'(a)^T u = 0 \Rightarrow u^T(F(x) - F(a)) = 0$. This implication is equivalent to the inclusion that $N[F'(a)^T] \subset N[F(x) - F(a)]$. Note that this null space condition is trivially satisfied, in particular, when $F'(a)$ is *onto*.

THEOREM 4.1. *Consider the problem* (PF). *Let a be a feasible point of* (PF). *Suppose that the constraint qualification in Corollary* 3.1 *holds. Assume that for each feasible x of* (PF), $N[F'(a)^T] \subset N[F(x)-F(a)]$. *Then, a is a global minimum for* (PF) *if and only if there exist* $\lambda_i \geqq 0$, $i=1,2,\cdots,m$ *and* $v_i \in \partial f_i(F(a))$, $i=0,1,2,\cdots,m$ *such that*

$$v_0^T F'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F'(a) = 0, \qquad \lambda_i f_i(F(a)) = 0.$$

Proof. The necessary conditions follow from Corollary 3.1. We shall prove the **sufficient part.** Let $x \in X$ be a feasible point of (PF). Then, from the convexity property of f, we get,

$$f_0(F(x)) - f_0(F(a)) \geqq v_0^T(F(x)-F(a)).$$

From the null space condition, there exists $\mu(x,a) \in X$ such that $F(x)-F(a) = F'(a)\mu(x,a)$. Now,

$$\begin{aligned} f_0(F(x)) - f_0(F(a)) &\geqq v_0^T F'(a)\mu(x,a), \\ &= -\sum_{i=1}^{m} \lambda_i v_i^T F'(a)\mu(x,a), \\ &\geqq -\sum_{i=1}^{m} \lambda_i f_i(F(x)) + \sum_{i=1}^{m} \lambda_i f_i(F(a)), \\ &\geqq 0, \end{aligned}$$

since x is feasible, f_i is convex, and $\lambda_i f_i(F(a)) = 0$. Hence, $f_0(F(x)) \geqq f_0(F(a))$, for each feasible $x \in X$ and so a is a global minimum of (PF). □

Theorem 4.1 extends the well-known necessary and sufficient optimality theorem for convex programming problems (e.g., [6]) to a class of convex composite non-differentiable problems. A nonsmooth version of the Wolfe duality theorem [6] is now presented for the problem (PF). Using the optimality conditions in the previous theorem, the following dual problem for (PF) can now be formulated:

(DF)
$$\begin{aligned} &\text{maximize} && f_0(F(\zeta)) + \sum_{i=1}^{m} \lambda_i f_i(F(\zeta)) \\ &\text{subject to} && 0 \in F'(\zeta)^T \partial f_0(F(\zeta)) + \sum_{i=1}^{m} \lambda_i F'(\zeta)^T \partial f_i(F(\zeta)), \quad \lambda \geqq 0. \end{aligned}$$

Note that the generalized subdifferential of the convex composite function $f_i \circ F$ is the set $\partial^{\diamond}(f_i \circ F)(\zeta) := F'(\zeta)^T \partial f_i(F(\zeta))$. Note also that $z \in F'(\zeta)^T \partial f_i(F(\zeta))$ if and only if $z = v_i^T F'(\zeta)$, for some $v_i \in \partial f_i(F(\zeta))$.

THEOREM 4.2 (duality). *For the problem* (PF), *let* f_i, $i=0,1,2,\cdots,m$, *be convex, let F be locally Lipschitz and Gâteaux differentiable on X, let* (PF) *attain a minimum at* $x=a$, *and let the Kuhn-Tucker type conditions* (3.5) *hold at a. If for each feasible* (ζ,λ) *of* (DF), *and x of* (PF), $N[F'(\zeta)^T] \subset N[F(x)-F(\zeta)]$, *then the dual problem* (DF) *attains its maximum and the optimal values of* (PF) *and* (DF) *are equal.*

Proof. Let x be feasible for (PF) and let $(\zeta,\lambda) \in X \times R^m$ be feasible for (DF). Then, there exist $v_i \in \partial f_i(F(\zeta))$, $i=0,1,2,\cdots,m$, such that $v_0^T F'(\zeta) + \sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta) = 0$. So,

$$\begin{aligned} &f_0(F(x)) - \left[f_0(F(\zeta)) + \sum_{i=1}^{m} \lambda_i f_i(F(\zeta)) \right] \\ &\quad \geqq v_0^T(F(x)-F(\zeta)) - \sum_{i=1}^{m} \lambda_i f_i(F(\zeta)) \quad \text{(by convexity of } f_0\text{)} \end{aligned}$$

$$= v_0^T F'(\zeta)\psi(x,\zeta) - \sum_{i=1}^{m} \lambda_i f_i(F(\zeta)) \quad \text{for some } \psi(x,\zeta) \in X$$

(by the null space condition)

$$= -\sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta)\psi(x,\zeta) - \sum_{i=1}^{m} \lambda_i f_i(F(\zeta)) \quad \text{(by feasibility)}$$

$$\geqq -\sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta)\psi(x,\zeta) - \sum_{i=1}^{m} \lambda_i f_i(F(x)) + \sum_{i=1}^{m} \lambda_i v_i^T (F(x) - F(\zeta))$$

(by convexity of f_i and nonnegativity of λ_i)

$$\geqq -\sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta)\psi(x,\zeta) + \sum_{i=1}^{m} \lambda_i v_i^T (F(x) - F(\zeta)) \quad \text{(by feasibility)}$$

$$= -\sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta)\psi(x,\zeta) + \sum_{i=1}^{m} \lambda_i v_i^T F'(\zeta)\psi(x,\zeta)$$

$$= 0.$$

Hence, $f_0(F(x)) \geqq f_0(F(\zeta)) + \sum_{i=1}^{m} \lambda_i f_i(F(\zeta))$; thus the weak duality holds.

Now, from the optimality condition (3.5), there exists $\lambda \in R^m$, $\lambda \geqq 0$ such that (a, λ) is feasible for (DF) and $\sum_{i=1}^{m} \lambda_i f_i(F(a)) = 0$. So,

$$\max (\text{DF}) \geqq f_0(F(a)) + \sum_{i=1}^{m} \lambda_i f_i(F(a)) = \min (\text{PF}).$$

This with the weak duality shows that (a, λ) is optimal for (DF) and the optimal values are equal. □

5. Convex transformable composite programs. In this section, we consider the following convex composite problem:

$$\text{(PFT)} \qquad \begin{aligned} &\text{minimize} && f_0(F(x)) \\ &\text{subject to} && x \in X, \quad f_i(F(x)) \leqq 0, \quad i = 1, 2, \cdots, m, \end{aligned}$$

where f_i, $i = 0, \cdots, m$, are convex functions and $F: X \to R^n$ is a locally Lipschitz and Gâteaux differentiable function with **range** of F, $F(X)$, **convex.** We shall see that these model problems (PFT) provide a class of programs which are not convex but possess some of the nice properties that convex programs have.

Following Heal [9], these programs are termed *convex transformable programs.* However, our assumptions are much weaker than the ones used in [9, p. 402], where F is assumed to be a bijective mapping. For various properties of differentiable convex transformable programs, see [8] and [9]. The value function from R^m to $[-\infty, \infty]$ for (PFT) is defined by

$$V(z) := \inf \{ f_0(F(x)) \mid x \in X, f_i(F(x)) \leqq z_i, \ i = 1, 2, \cdots, m \}.$$

Note that infimum over the empty set is $+\infty$. The domain of V is denoted by

$$\text{dom } (V) = \{ z \in R^m \mid V(z) < +\infty \}.$$

We first observe the useful property that the *value function* of the convex composite problem (PFT) is *convex.* This follows easily from a known result of convex programming [18], and [19] by writing the value function $V(z)$ as

$$V(z) = \inf \{ f_0(y) \mid y \in F(X), f_i(y) \leqq z_i, \ i = 1, 2, \cdots, m \}.$$

Note here that $F(X)$ is convex and, for each $i = 1, 2, \cdots, m$, f_i is convex. We further note that, although (PFT) is not a convex program, it falls into the category of *convexlike* program in the following sense:

$$(\forall \alpha \in (0,1)) \quad (\forall x_1, x_2 \in X) \quad (\exists x_0 \in X)$$
$$f_i(F(x_0)) \leqq \alpha f_i(F(x_1)) + (1-\alpha) f_i(F(x_2)),$$

for each $i = 0, 1, 2, \cdots, m$. For detailed study of this class of programs, see [12] and [15].

It is well known that, for a convex program, the Lagrange multipliers in the Kuhn-Tucker relations can be completely characterized in terms of subgradients of the value function. Here we show that this characterization is not limited to convex problems, but continues to hold for the class of nonsmooth (nonconvex) composite problems (PFT). The convexity property of the value function and the composite structure in the program (PFT) allow us to obtain such a characterization theorem for Lagrange multipliers in terms of subgradients of the value function. The interior of a convex set C is denoted by int C. The Lagrangian function for (PFT) is denoted by $L(x, \lambda) = f_0(F(x)) + \sum_{i=1}^{m} \lambda_i f_i(F(x))$.

THEOREM 5.1. *For the problem* (PFT), *let* f_i, $i = 0, 1, \cdots, m$ *be convex*; *let* $F: X \to R^n$ *be locally Lipschitz and Gâteaux differentiable on* X *with* $F(X)$ *convex. Suppose that* $0 \in \text{int}(\text{dom}(V))$. *If* (PFT) *attains its minimum at* $x = a$ *and if* $F'(a)$ *is onto then the value function* V *is continuous at* 0, $\partial V(0)$ *is nonempty and*

$$\partial V(0) = \left\{ -\lambda \;\middle|\; \lambda \in R^m, \lambda \geqq 0, v_0^T F'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F'(a) = 0, \right.$$
$$\left. v_i \in \partial f_i(F(a)), \lambda_i f_i(F(a)) = 0, \quad i = 1, 2, \cdots, m \right\}.$$

Proof. Since the value function V is convex, the first two conclusions of the theorem follow from the known results of convex programming [19, Thms. 16-18]. Now, from the Lagrangian duality theorem in [12], there exists $\lambda \in R^m$, $\lambda \geqq 0$ such that

$$\min_{x \in X} L(x, \lambda) = f_0(F(a)) \quad \text{and} \quad \lambda_i f_i(F(a)) = 0,$$

since (PFT) is a convexlike program. Furthermore,

$$\partial V(0) = \left\{ -\lambda \;\middle|\; \lambda \in R^m, \lambda \geqq 0, f_0(F(a)) = \min_{x \in X} L(x, \lambda) \right\}.$$

The last conclusion will follow from this if we show that $\min_{x \in X} L(x, \lambda) = f_0(F(a))$ if and only if $v_0^T F'(a) + \sum_{i=1}^{m} \lambda_i v_i^T F'(a) = 0$, for some $v_i \in \partial f_i(F(a))$. Since $\lambda_i f_i(F(a)) = 0$, the function $f_0 + \sum_{i=1}^{m} \lambda_i f_i$ is convex and $F'(a)$ is onto, the equivalence follows from Proposition 4.2, and hence the proof is completed. □

Remark 5.1. It should be noted that if $0 \in \text{int}(\text{dom}(V))$ then the generalized Slater condition [12], that $f_i(F(x_0)) < 0$, $i = 1, 2, \cdots, m$, for some $x_0 \in X$, holds. Moreover, it is worth noting that Theorem 5.1 may also be derived from certain results in convex analysis by converting the problem (PFT) into an equivalent convex problem.

Acknowledgments. The author is extremely grateful to the referees for their valuable suggestions and helpful comments, which have contributed to the final preparation of this paper.

REFERENCES

[1] J. R. BIRGE AND L. QI, *The Michel-Penot Subdifferential and Stochastic Programming*, Applied Mathematics Preprint, No. 12, University of New South Wales, Kensington, New South Wales, Australia, 1989.

[2] J. M. BORWEIN, S. P. FITZPATRICK, AND J. R. GILES, *The differentiability of real functions on normed linear space using generalized subgradients*, J. Math. Anal. Appl., 128 (1987), pp. 512–534.
[3] J. V. BURKE, *Descent methods for composite nondifferentiable optimization problems*, Math. Programming, 33 (1985), pp. 260–179.
[4] R. W. CHANEY, *On sufficient conditions in nonsmooth optimization*, Math. Oper. Res., 7 (1982), pp. 463–475.
[5] F. H. CLARKE, *Optimization and Nonsmooth Analysis*, John Wiley, New York, 1983.
[6] B. D. CRAVEN, *Mathematical Programming and Control Theory*, Chapman and Hall, London, 1978.
[7] R. FLETCHER, *Practical Methods of Optimization*, John Wiley, New York, 1987.
[8] M. A. HANSON AND B. MOND, *Convex transformable programming problems and invexity*, J. Inform. Optim. Sci., 8 (1987), pp. 201–207.
[9] G. HEAL, *Equivalence of saddle-points and optima for non-concave programs*, Adv. in Appl. Math., 5 (1984), pp. 398–415.
[10] J. B. HIRIART-URRUTY, *Mean value theorems in nonsmooth analysis*, Numer. Funct. Anal. Optim., 2 (1980), pp. 1–30.
[11] A. D. IOFFE, *Necessary and sufficient conditions for a local minimum* 2: *Conditions of Levitin–Miljutin–Osmolovskii type*, SIAM J. Control Optim., 17 (1979), pp. 251–265.
[12] V. JEYAKUMAR, *Convexlike alternative theorems and mathematical programming*, Optimization, 16 (1985), pp. 643–652.
[13] ———, *On optimality conditions in nonsmooth inequality constrained minimization*, Numer. Funct. Anal. Optim., 9 (1987), pp. 535–546.
[14] ———, *Generalized differentiability properties of locally Lipschitz functions*, in preparation.
[15] V. JEYAKUMAR AND H. WOLKOWICZ, *Zero duality gaps in infinite dimensional programming*, Research Report 24, University of New South Wales, New South Wales, Australia, October 1988; J. Optim. Theory Appl., 67 (1990), pp. 87–108.
[16] P. MICHEL AND J. P. PENOT, *Calcul sous-differentiel pour des fonctions lipschitziennes et non lipschitziennes*, Comptes Rendus de l'Academie des Sciences Paris, 298 (1984), pp. 269–272.
[17] B. N. PSHENICHNYI, *Necessary Conditions for an Extremum*, Marcel Dekker, New York, 1971.
[18] R. T. ROCKAFELLAR, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1969.
[19] ———, *Conjugate Duality and Optimization*, CBMS-NSF Regional Conference Series in Applied Mathematics, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1974.
[20] ———, *Generalized directional derivatives and subgradients of nonconvex functions*, Canad. J. Math., 32 (1980), pp. 257–280.
[21] ———, *First and second order epi-differentiability in nonlinear programming*, Trans. Amer. Math. Soc., 307 (1988), pp. 75–108.
[22] M. STUDNIARSKI, *Mean value theorems and sufficient optimality conditions for nonsmooth functions*, J. Math. Anal. Appl., 111 (1985), pp. 313–326.
[23] D. E. WARD, *Convex subcones of the contingent cone in nonsmooth calculus and optimization*, Trans. Amer. Math. Soc., 302 (1987), pp. 661–682.
[24] D. E. WARD AND J. M. BORWEIN, *Nonsmooth calculus in finite dimensions*, SIAM J. Control Optim., 25 (1987), pp. 1304–1312.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 42-56, February 1991

005

LOCAL AND SUPERLINEAR CONVERGENCE FOR PARTIALLY KNOWN QUASI-NEWTON METHODS*

JOHN R. ENGELS† AND HÉCTOR J. MARTÍNEZ†‡

Abstract. This paper develops a unified theory for establishing the local and q-superlinear convergence of quasi-Newton methods from the convex class when part of the Hessian matrix is known. One first proves the bounded deterioration principle due to Dennis (and popularized by Broyden, Dennis, and Moré) for the appropriate modifications of all update formulas in the convex Broyden class. Using standard conditions on the quasi-Newton updates, one then deduces local and q-superlinear convergence. Particular cases of these methods are the SQP augmented scale BFGS and DFP secant methods for constrained optimization problems introduced by Tapia and a generalization of the Al-Baali and Fletcher modification of the structured secant method considered by Dennis, Gay, and Welsch for the nonlinear least-squares problem. In all cases, bounded deterioration is proved for the approximate Hessian, not for its inverse.

Key words. convergence theory, bounded deterioration, superlinear convergence, unconstrained optimization, constrained optimization, least squares, secant methods, quasi-Newton methods

AMS(MOS) subject classifications. 65K05, 90C30

1. Introduction.

In this paper we extend the results for the partially known BFGS secant method of Dennis, Martínez, and Tapia [11] and prove convergence theorems for those secant methods which use a secant update from the convex class studied by Broyden [4] and Fletcher [17] and have been modified to take advantage of a known part of the Hessian in constructing approximate Hessians.

The theory we will present can also be viewed as a generalization of the result for the partially known DFP secant method given by Dennis and Walker [14] to any partially known secant method with updates in the convex class. It is an extension of the results for secant methods which use updates from Broyden's convex class obtained by Griewank and Toint [19] to handle those partially known secant methods which use a secant update in the same class. Indeed, our approach is similar to the ones used in all three of these papers. We follow Griewank and Toint in showing bounded deterioration for the Hessian approximations in every case rather than for their inverses.

The methods of interest in this paper are iterative methods for solving

$$\underset{x\in\mathbf{R}^n}{\text{minimize}} f(x), \quad f:\mathbf{R}^n\to\mathbf{R}$$

in the case when the complete computation of the Hessian matrix $\nabla^2 f(x)$ is infeasible. In several applications, the Hessian matrix is partially available; this suggests that we work with an approximation B_k of $\nabla^2 f(x_k)$ of the form

$$\nabla^2 f(x_k) \simeq B_k = C(x_k) + A_k$$

where $C(x_k)$ is a computed part of $\nabla^2 f(x_k)$ and A_k is an approximated part of $\nabla^2 f(x_k)$. For example, such approximations are used in the nonlinear least-squares case [1], [10] and proved to be very competitive, at least in the large residual case. For more details on partially known updates and an historical survey see Dennis, Martínez, and Tapia [11] or Martínez [20].

* Received by the editors December 29, 1989; accepted for publication February 22, 1990.

† Department of Mathematics, Facultés Universitaires ND de la Paix, B-5000 Namur, Belgium.

‡ The research of this author was sponsored by SDIO/IST/ARO, Air Force Office of Scientific Research grant 85-0243, U.S. Department of Energy grant DEFG05-86ER 25017, Department of Mathematical Sciences, Rice University, Houston, Texas 77251-1982, and by COLCIENCIAS, Departamento de Matemáticas, Universidad del Valle, A.A. 25360, Cali, Colombia.

But nonlinear least squares is not the only case where algorithms are of interest. Engels [15] used an algorithm of reduced gradient type to solve optimization problems for large scale econometric models. He used the same framework to solve the reduced problem, where $C(x)$ is the known part of the reduced Hessian. His tests outperformed a simple BFGS algorithm by a factor of 2 to 30, which increased with the size of the problem. Another important application of partially known secant methods was given by Tapia [22]. He used the well-known bounded deterioration of the DFP and the inverse form of the BFGS secant updates as a basis for establishing bounded deterioration of the partially known DFP and the inverse of the partially known BFGS secant updates. Then he proved local and q-superlinear convergence for the partially known DFP and BFGS secant versions of his algorithms for equality-constrained optimization problems. We will give more details about these algorithms in § 4.

We will call the class of symmetric rank-2 secant updates suggested by Broyden [4] *the Broyden class of secant updates.* Fletcher [17] shows that this class of secant updates can be written as

$$B_+ = \text{Broy}(s, y, B, \phi) \tag{1a}$$

where the parameter $\phi \in \mathbf{R}$, and $\text{Broy}(s, y, B, \phi)$ is given by

$$\text{Broy}(s, y, B, \phi) = B + \frac{yy^t}{y^t s} - \frac{Bss^tB}{s^tBs} + \phi s^tBs\, uu^t \tag{1b}$$

$$u = \frac{y}{y^t s} - \frac{Bs}{s^tBs}. \tag{1c}$$

The following are well-known choices of the parameter ϕ:

(2a)	Convex Class	$\phi \in [0, 1]$,
(2b)	DFP	$\phi = 1$,
(2c)	BFGS	$\phi = 0$,
(2d)	SR1	$\phi = \dfrac{y^t s}{y^t s - s^tBs}$.

Assuming that

$$\nabla^2 f(x) = C(x) + S(x) \tag{3}$$

where $C: \mathbf{R}^n \to \mathbf{R}^{n\times n}$ is the available part of $\nabla^2 f$, we consider the following class of algorithms: For given x_0 and A_0 symmetric, define

$$\begin{aligned} B_k &= C(x_k) + A_k, \\ x_{k+1} &= x_k - B_k^{-1}\nabla f(x_k), \\ A_{k+1} &\in U(x_k, A_k), \end{aligned} \tag{4}$$

where the update function U will be defined later (see (7) and (8) in § 2).

Throughout this paper we make the following *standard assumptions.*

Let $f: \mathbf{R}^n \to \mathbf{R}$ be two times differentiable in an open convex neighborhood Ω of a strong local minimizer x_* with $\nabla^2 f(x_*)$ positive definite, and let $C: \mathbf{R}^n \to \mathbf{R}^{n\times n}$ be such that $C(x)$ is symmetric. Assume further that $\nabla^2 f$ and C are locally Hölder continuous at x_* and let $\nu \geqq 0$, $\nu_c \geqq 0$ and $p \in]0, 1]$ be such that for all $x \in \Omega$

$$|\nabla^2 f(x) - \nabla^2 f(x_*)| \leqq \nu |x - x_*|^p \quad \text{and} \quad |C(x) - C(x_*)| \leqq \nu_c |x - x_*|^p$$

where $|\cdot|$ denotes a vector norm and its subordinate operator norm.

In addition, we will use the following standard notation:

$$s_k = x_{k+1} - x_k,$$

$$y_k \approx \nabla^2 f(x_*) s_k,$$

$$\sigma(x_1, x_2) = \max(|x_1 - x_*|^p, |x_2 - x_*|^p),$$

$$\sigma_k = \sigma(x_{k+1}, x_k),$$

$$\Sigma(x_1, x_2) = \{x \,|\, |x - x_*| \leqq \max(|x_1 - x_*|, |x_2 - x_*|)\},$$

$$\Sigma_k = \Sigma(x_{k+1}, x_k).$$

We will omit the subscript k, replace the subscript $k+1$ by the subscript $+$, and use f_k or C_* instead of $f(x_k)$ or $C(x_*)$ when no ambiguity is possible. One way of defining y, and the most often used, is

$$y = \nabla f(x_+) - \nabla f(x),$$

but it can also be defined as

$$y = y^{\#} + C(\bar{x})s$$

where $\bar{x} \in \Sigma_k$ and $y^{\#}$ is a given approximation of $S(x_*)s$.

One of the main issues in the development of the theory given in this paper is the bounded deterioration principle given by Dennis [7] and popularized by Broyden, Dennis, and Moré [5]. It was Griewank and Toint [19] who first established a unified bounded deterioration principle for all the members of the convex class ((1) with (2a)). In the same paper, they gave sufficient conditions for a member of this subclass of secant methods to have a q-superlinear rate of convergence. However, mainly due to the fact that they assume that the problem had been transformed into a particular form and their *big O* notation, it was not obvious how to extend this result to the partially known secant methods described above. It also was not clear how to obtain the direct form rather than the inverse form of the bounded deterioration principle for the partially known secant methods, except DFP, from other approaches in the literature. The theory given in §§ 2 and 3 answers the open question of the local and q-superlinear convergence of any partially known secant method which uses a secant update from the convex class.

In § 2 we prove that the partially known secant approximations to the Hessian defined in § 1 satisfy the bounded deterioration principle for $\phi \in [0, 1]$. Moreover, by just setting $\phi = 0$ we will have the surprising and stronger form of this bounded deterioration for the partially known BFGS secant method given by Dennis, Martínez, and Tapia [11].

In § 3 we establish the local and q-superlinear convergence for all of the partially known secant methods which use updates from the convex class using the theories of Broyden, Dennis, and Moré [5], Griewank and Toint [19], and Dennis and Moré [12]

Finally, in § 4 we use this theory to prove the local and q-superlinear convergence of any partially known secant method which uses a secant update from the convex class for the equality-constrained optimization problem and the nonlinear least-squares problem. Particular cases of these methods are the SQP augmented scale BFGS and DFP secant methods for constrained optimization problems introduced by Tapia [22]. Another particular case, for which local and q-superlinear convergence was proved for the first time in Dennis, Martínez, and Tapia [11], is the Al-Baali and Fletcher [1] modification of the partially known BFGS secant method considered by Dennis, Gay,

and Welsch [10] for the nonlinear least-squares problem and implemented in the current version of the NL2SOL code.

2. Bounded deterioration. Our objective in this section is to show that the partially known quasi-Newton approximations to the Hessian derived from the convex Broyden class ((1) with (2a)) satisfy the direct form of the bounded deterioration principle, i.e., for x sufficiently close to x_*, these updates satisfy

$$\|B_+ - \nabla^2 f(x_*)\| \leqq [1+\alpha_1\sigma(x, x_+)]\|B - \nabla^2 f(x_*)\| + \alpha_2\sigma(x, x_+) \tag{5}$$

where α_1 and α_2 are positive constants, and $\|\cdot\|$ is the Frobenius norm weighted by $\nabla^2 f(x_*)$.

The bounds needed to prove inequality (5) follow from the standard assumptions and the fact that y and $y^\#$ are "good" approximations to $\nabla^2 f(x_*)s$ and $S(x_*)s$, respectively. We formalize this in the following two propositions.

PROPOSITION 1. *Let f satisfy the standard assumptions, let D be a neighborhood of x_*, and let y be an approximation to $\nabla^2 f(x_*)s$. Assume that there exists a constant κ_1 such that for each x_1, $x_2 \in D$ and $s = x_2 - x_1$ it holds that*

$$|y - \nabla^2 f(x_*)s| \leqq \kappa_1\sigma(x_1, x_2)|s|.$$

Then there are positive constants ε, κ_2, κ_3, κ_4, κ_5, and κ_6 such that for all x_1, $x_2 \in D_1$, defined by $D_1 = \{x \,|\, |x - x_*| \leqq \varepsilon\} \subset D$, the following inequalities hold:

$$|y|_2 \leqq (\kappa_2 + \kappa_3\sigma(x_1, x_2))|s|_2, \tag{6a}$$

$$y^t s \leqq (\kappa_2 + \kappa_3\sigma(x_1, x_2))|s|_2^2, \tag{6b}$$

$$y^t s \geqq \kappa_4|s|_2^2, \tag{6c}$$

$$\frac{|y|_2|s|_2}{y^t s} \leqq \kappa_5 + \kappa_6\sigma(x_1, x_2), \qquad s \neq 0. \tag{6d}$$

Proof. We will make strong use of the equivalence of all norms on $\mathbf{R}^n$; in particular, for $|\cdot|$ and $|\cdot|_2$ (the Euclidean norm) we assume for some ξ_1 and ξ_2 and any $x \in \mathbf{R}^n$ that

$$\frac{1}{\xi_1}|x|_2 \leqq |x| \leqq \xi_2|x|_2.$$

Let x_1, $x_2 \in D$ and let β_1 and β_2 be constants such that

$$|\nabla^2 f^{-1}(x_*)| \leqq \beta_1, \qquad |\nabla^2 f(x_*)| \leqq \beta_2;$$

then (6a) and (6b) follow directly from the fact that $y = (y - \nabla^2 f(x_*)s) + \nabla^2 f(x_*)s$ with $\kappa_2 = \xi_1\xi_2\kappa_1$ and $\kappa_3 = \xi_1\xi_2\beta_2$. Now choose ε so that $\xi_1^2\xi_2^2\kappa_1\varepsilon^p\beta_1 \leqq 1$ and $D_1 \subset D$; then (6c) follows immediately with $\kappa_4 = (1 - \xi_1^2\xi_2^2\kappa_1\varepsilon^p\beta_1)/\xi_1\xi_2\beta_1$. Finally, notice that for $s \neq 0$

$$\frac{|y|_2|s|_2}{y^t s} = \frac{|y|_2}{|s|_2}\frac{|s|_2^2}{y^t s},$$

so that (6d) follows from inequalities (6a) and (6c) with $\kappa_5 = \kappa_2/\kappa_4$ and $\kappa_6 = \kappa_3/\kappa_4$. □

PROPOSITION 2. *Let f and C satisfy the standard assumptions, let $D \subset \Omega$ be a neighborhood of x_*, and let $y^\#$ be an approximation to $S(x_*)s$. Assume that there exists a constant κ_7 such that for each x_1, $x_2 \in D$ and $s = x_2 - x_1$ it holds that*

$$|y^\# - S(x_*)s| \leqq \kappa_7\sigma(x_1, x_2)|s|.$$

Then there exists a positive constant κ_1 *such that for all* $x_1, x_2 \in D$, $y = y^{\#} + C(\bar{x})s$ *satisfies*

$$|y - \nabla^2 f(x_*)s| \leqq \kappa_1 \sigma(x_1, x_2)|s|$$

whenever $\bar{x} \in \Sigma(x_1, x_2)$.

Proof. Let $x_1, x_2 \in D$. Taking advantage of the structure in y and in the Hessian (3), we can write

$$\begin{aligned} |y - \nabla^2 f(x_*)s| &\leqq |y^{\#} - S(x_*)s| + |[C(\bar{x}) - C(x_*)]s| \\ &\leqq \kappa_7 \sigma(x_1, x_2)|s| + \nu_c |\bar{x} - x_*|^p |s| \\ &\leqq (\kappa_7 + \nu_c)\sigma(x_1, x_2)|s|. \end{aligned}$$

□

Before establishing the bounded deterioration inequality for the partially known quasi-Newton updates derived from the convex class, we reformulate the corresponding result obtained by Griewank and Toint [19] for the convex class. Our hypotheses are slightly weaker than those used by Griewank and Toint. This will permit us later to generalize their result to the partially known quasi-Newton updates.

PROPOSITION 3. *Assume that f satisfies the standard assumptions and let* B_+ *be a secant update from the convex class* ((1) *with* (2a)) *where* $s = x_+ - x$ *and y is an approximation to* $\nabla^2 f(x_*)s$. *If there exist* $D \subset \Omega$ *a neighborhood of* x_* *and a constant* κ_1 *such that for each x,* $x_+ \in D$ *it holds that*

$$|y - \nabla^2 f(x_*)s| \leqq \kappa_1 \sigma(x, x_+)|s|.$$

Then there exists a neighborhood $D_1 \subset D$ *such that the bounded deterioration inequality* (5) *holds whenever* $x, x_+ \in D_1$ *and B is positive definite.*

Proof. In their paper Griewank and Toint assume that $y = \nabla^2 f(x_+) - \nabla^2 f(x)$, but to prove the bounded deterioration for the convex class they only need inequalities like (6). Therefore their proof remains correct in this case and the result follows from Proposition 1. Moreover, using the same set of inequalities and a slightly different approach, it is possible to prove that inequality (5) holds with $\alpha_1 = \phi\rho_0$ and $\alpha_2 = \rho_1 + \phi\rho_0$ for some positive constants ρ_0 and ρ_1 (see [20, Thm. 2.4] for more details). □

Note that the stronger form of the bounded deterioration, i.e., $\alpha_1 = 0$ when $\phi = 0$, obtained by Dennis, Martínez, and Tapia [11] for the BFGS secant update follows from the last observation.

We will now prove an analogous result for the partially known quasi-Newton updates derived from the convex class. In the theorems hereafter we use an update function which is slightly more general than this class. We use this update function to clarify the fact that at each iteration we can change the update formula of the approximation of the Hessian matrix without losing any of the convergence properties stated hereafter. Let us define the following update functions

$$U^1(x, A) = \bigcup_{\bar{x} \in \Sigma(x, x_+)} \{A + C(\bar{x}) - C_+\}, \tag{7a}$$

$$U^2(x, A) = \bigcup_{\bar{x} \in \Sigma(x, x_+)} \bigcup_{\phi \in [0, 1]} \{\text{Broy}(s, y, C(\bar{x}) + A, \phi) - C_+\}, \tag{7b}$$

$$U(x, A) = U^1(x, A) \cup U^2(x, A) \tag{7c}$$

with

$$\begin{aligned} &s = x_+ - x, \qquad y = y^{\#} + C(\bar{x})s, \\ &y^{\#} \text{ is an approximation to } S(x_*)s. \end{aligned} \tag{7d}$$

First note that these update functions of A correspond to the following update functions

of B:

$$U_B^1(x, B) = \bigcup_{\bar{x}\in\Sigma(x, x_+)} \{B + C(\bar{x}) - C\}, \tag{8a}$$

$$U_B^2(x, B) = \bigcup_{\bar{x}\in\Sigma(x, x_+)} \bigcup_{\phi\in[0, 1]} \{\text{Broy}\,(s, y, C(\bar{x}) + A, \phi)\}, \tag{8b}$$

$$U_B(x, B) = U_B^1(x, B) \cup U_B^2(x, B), \tag{8c}$$

and note that taking $U^1(x, A)$ with $\bar{x} = x_+$ corresponds to an update formula in which only the computed part C is updated and the matrix A remains constant; taking $U^1(x, A)$ with $\bar{x} = x$ corresponds to the classical chord method, i.e., the matrix B remains constant; taking $U^2(x, A)$ with $\bar{x} = x_+$ corresponds to the most common partially known quasi-Newton updates derived from the convex class, and taking $U^2(x, A)$ with $\bar{x} = x$ corresponds to the standard convex class.

Let us now state our first result.

THEOREM 4. *Let f and C satisfy the standard assumptions and let B_+ be an update of B defined by*

$$B_+ = A_+ + C_+$$

where $A_+ \in U(x, A)$, given by (7), *and $x_+ = x - B^{-1}\nabla f(x)$. If there exist $D \subset \Omega$, a neighborhood of x_*, and a constant κ_7 such that for each x, $x_+ \in D$ it holds that*

$$|y^\# - S(x_*)s| \leqq \kappa_7 \sigma(x, x_+)|s|.$$

Then there exists a neighborhood $N = N_1 \times N_2$ of $(x_*, \nabla^2 f(x_*))$ such that the bounded deterioration inequality (5) *holds whenever $(x, B) \in N$.*

Proof. From Propositions 2 and 3 we can easily deduce that there exist positive constants ξ_1 and ξ_2 such that $B_+ = \text{Broy}\,(s, y, B, \phi)$ with $\phi \in [0, 1]$ satisfies

$$\|B_+ - \nabla^2 f(x_*)\| \leqq (1 + \xi_1\sigma)\|B - \nabla^2 f(x_*)\| + \xi_2\sigma \tag{9}$$

whenever $x, x_+ \in D_1$ and B is positive definite.

We will now prove a similar result for $U_B(x, B)$. We will make strong use of the equivalence of all norms on $\mathbf{R}^{n\times n}$; in particular, for $|\cdot|$ and $\|\cdot\|$ we assume for some $\eta > 0$ and any $M \in \mathbf{R}^{n\times n}$, that

$$\|M\| \leqq \eta|M|. \tag{10}$$

If we define the neighborhoods N_1 of x_* and N_2 of $\nabla^2 f(x_*)$ as

$$N_1 = \{x \,|\, |x - x_*| < \varepsilon_1\}$$

and

$$N_2 = \{B \,|\, B = A + C(x),\ x \in N_1 \text{ and } |A - S(x_*)| < \varepsilon_2\}$$

then, following the same arguments used by Broyden, Dennis, and Moré [5], it is possible to prove that there exist constants ε_1 and ε_2 such that $N_1 \subset D_1$, N_2 contains only positive-definite matrices and $x_+ \in N_1$ for any $x \in N_1$ and $B \in N_2$ (see Engels [16, Thm. 1] for more details).

It follows that $\bar{M} = A + \bar{C} = B + \bar{C} - C$, is positive definite and $\bar{M}^{-1}$ well defined. Therefore $\text{Broy}\,(s, y, B, \phi)$ and $\text{Broy}\,(s, y, \bar{M}, \phi)$ with $\phi \in [0, 1]$ and $\bar{x} \in \Sigma(x, x_+)$ are well defined and satisfy (9).

Now, since

$$\begin{aligned} \|\bar{M} - \nabla^2 f_*\| &= \|\bar{C} + A - \nabla^2 f_* + C_* - C_* + C - C\| \\ &= \|B - \nabla^2 f_* + (\bar{C} - C_*) + (C_* - C)\| \\ &\leqq \|B - \nabla^2 f_*\| + 2\eta\nu_c\sigma, \end{aligned} \tag{11}$$

inequality (5) holds with $\alpha_1 = 0$ and $\alpha_2 = 2\eta\nu_c$ for $B_+ \in U_B^1(x, B)$ and, if $B_+ \in U_B^2(x, B)$, we have from (9)-(11)

$$\begin{aligned}\|B_+ - \nabla^2 f_*\| &\leqq (1+\xi_1\sigma)\|\bar{M} - \nabla^2 f_*\| + \xi_2\sigma \\ &\leqq (1+\xi_1\sigma)\|B - \nabla^2 f_*\| + \xi_3\sigma,\end{aligned}$$

with $\xi_3 \geqq (\xi_2 + 2\xi_1\varepsilon_1^p\eta\nu_c + 2\eta\nu_c)$. We conclude that for each $(x, B) \in N = N_1 \times N_2$, and $x_+ = x - B^{-1}\nabla f(x)$, the bounded deterioration inequality (5) holds for each $B_+ \in U_B(x, B)$. □

3. Local convergence theory. In this section we will establish the local and q-superlinear convergence of the partially known quasi-Newton updates derived from the convex class. The local and linear convergence follows directly from the results of § 2 and the Broyden–Dennis–Moré theory. For completeness we restate the linear convergence theorem as follows.

THEOREM 5. *Let f and C satisfy the standard assumptions. Then for each $r \in]0, 1[$, there are positive constants ε_r and δ_r such that for $|x_0 - x_*| < \varepsilon_r$ and $|A_0 - S(x_*)| < \delta_r$, the sequence $\{x_k\}$, defined by* (4) *with $U(x_k, A_k)$ defined by* (7), *is well defined and converges to x_*. Furthermore, it holds that*

$$|x_{k+1} - x_*| \leqq r|x_k - x_*|$$

for each $k \geqq 0$, and $\{|B_k|\}$, $\{|B_k^{-1}|\}$ are uniformly bounded.

Of course to prove the superlinear convergence we must restrict the update functions to their second part, i.e., $U^2(x, A)$ and $U_B^2(x, B)$, respectively. Due to the invariance properties of the updates in the Broyden family and the fact that C can be changed according to the changes in $\nabla^2 f$, we assume without loss of generality that

$$\nabla^2 f(x_*) = I.$$

The norm $\|\cdot\|$ then reduces simply to the Frobenius norm.

THEOREM 6. *Let f and C satisfy the standard assumptions mentioned in* § 1. *Then for (x_0, A_0) sufficiently close to $(x_*, S(x_*))$ the sequence $\{x_k\}$ defined by* (4) *with the update function defined by* (7b) *and* (7d), *converges q-superlinearly to x_*.*

Proof. We will prove that

$$\lim_k \frac{\|(B_k - I)s_k\|}{\|s_k\|} = 0. \tag{12}$$

The superlinear convergence then follows from the well-known Theorem 2.2 of Dennis and Moré [12].

First consider the matrix

$$B'_+ = \text{Broy}\,(s, s, \bar{M}, \phi)$$

where $\bar{M} = A + \bar{C} = B + \bar{C} - C$.

Using arguments similar to those of Griewank and Toint [19], we get

$$\begin{aligned} 0 \leqq h(\phi, \bar{M}, s) &\overset{\text{def}}{=} \|\bar{M} - I\|^2 - \|B'_+ - I\|^2 \\ &= (1-\phi)\left\{\left(1 - \frac{s^t\bar{M}\bar{M}s}{s^t\bar{M}s}\right)^2 + 2\left[\frac{s^t\bar{M}\bar{M}\bar{M}s}{s^t\bar{M}s} - \left(\frac{s^t\overline{MM}s}{s^t\bar{M}s}\right)^2\right]\right\} \\ &\quad + \phi\left\{\left(1 - \frac{s^t\bar{M}s}{s^ts}\right)^2 + 2\phi\left[\frac{s^t\bar{M}\bar{M}s}{s^ts} - \left(\frac{s^t\bar{M}s}{s^ts}\right)^2\right]\right\} \\ &\quad + \phi(1-\phi)\left\{\left(\frac{s^t\bar{M}\bar{M}s}{s^t\bar{M}s}\right)^2 - \left(\frac{s^t\bar{M}s}{s^ts}\right)^2\right\}. \end{aligned} \tag{13}$$

We can find a constant ξ_4 independent of ϕ, such that

$$\|B_+ - B'_+\| \leqq \xi_4(\|\bar{M} - I\| + 2)\sigma. \tag{14}$$

From Theorem 5, $\|B - I\|$ is bounded, and it holds that for some ξ_5

$$\begin{aligned}\|\bar{M} - I\| &\leqq \|\bar{M} - B\| + \|B - I\| \\ &\leqq 2\eta\nu_c\sigma + \|B - I\| \\ &\leqq \xi_5.\end{aligned}$$

It then follows from (14) that

$$\|B_+ - B'_+\| \leqq \xi_6\sigma$$

and consequently,

$$\begin{aligned}\|B_+ - I\| &\leqq \|B_+ - B'_+\| + \|B'_+ - I\| \\ &\leqq \|B'_+ - I\| + \xi_6\sigma, \\ \|B_+ - I\|^2 &\leqq (\|B'_+ - I\| + \xi_6\sigma)^2 \\ &\leqq \|B'_+ - I\|^2 + \sigma(2\xi_6\|B'_+ - I\| + \xi_6^2\sigma) \\ &\leqq \|B'_+ - I\|^2 + \sigma\xi_7, \\ -\|B'_+ - I\|^2 &\leqq -\|B_+ - I\|^2 + \sigma\xi_7\end{aligned} \tag{15}$$

for some ξ_6 and ξ_7. Using (13) we obtain that

$$\begin{aligned}0 \leqq h(\phi, \bar{M}, s) &= \|\bar{M} - I\|^2 - \|B'_+ - I\|^2 \\ &\leqq (\|\bar{M} - B\| + \|B - I\|)^2 - \|B'_+ - I\|^2 \\ &\leqq (2\eta\nu_c\sigma + \|B - I\|)^2 - \|B'_+ - I\|^2 \\ &\leqq \xi_8\sigma + \|B - I\|^2 - \|B'_+ - I\|^2\end{aligned} \tag{16}$$

for some ξ_8. We deduce from (15) and the linear convergence of $\{x_k\}$ (Theorem 5) that the sum

$$\begin{aligned}\sum_{k\geqq 0}(\xi_8\sigma_k + \|B_k - I\|^2 - \|B'_{k+1} - I\|^2) &\leqq \sum_{k\geqq 0}(\|B_k - I\|^2 - \|B_{k+1} - I\|^2 + (\xi_7 + \xi_8)\sigma_k) \\ &\leqq \|B_0 - I\|^2 + (\xi_7 + \xi_8)\sum_{k\geqq 0}\sigma_k\end{aligned}$$

is finite; and this implies that

$$\lim_k [\xi_8\sigma_k + \|B_k - I\|^2 - \|B'_{k+1} - I\|^2] = 0;$$

and by (16) it holds that

$$\lim_k h(\phi_k, \bar{M}_k, s_k) = 0.$$

Using arguments similar to those of Griewank and Toint [19], we prove, for an arbitrary sequence $\{\phi_k\}$ in $[0, 1]$ and an arbitrary sequence $\{\bar{x}_k\}$, $\bar{x}_k \in \Sigma_k$, that

$$\lim_k \frac{\|(\bar{M}_k - I)s_k\|}{\|s_k\|} = 0.$$

The conclusion (12) follows then from

$$0 \leqq \lim_k \frac{\|(B_k - I)s_k\|}{\|s_k\|} = \lim_k \frac{\|[(B_k + \bar{C}_k - C_k) - I + (C_* - \bar{C}_k) + (C_k - C_*)]s_k\|}{\|s_k\|}$$

$$\leqq \lim_k \frac{\|(\bar{M}_k - I)s_k\|}{\|s_k\|} + \lim_k 2\eta\nu_c\sigma_k$$

$$= 0. \qquad \square$$

4. Applications. In this section we use the results of §§ 2 and 3 to establish the local and q-superlinear convergence of any partially known secant method which uses an update from the convex class for the constrained optimization problem and the nonlinear least-squares problem. Particular cases of these methods are the SQP augmented scale BFGS and DFP secant methods for constrained optimization problems suggested by Tapia [22]. Another particular case (for which local and q-superlinear convergence was proved for the first time by Dennis, Martínez, and Tapia [11]) is the Al-Baali and Fletcher [1] modification of the partially known BFGS secant method considered by Dennis, Gay, and Welsch [10] for the nonlinear least-squares problem and implemented in the current version of the NL2SOL code.

4.1. Nonlinear least squares. Our presentation of the nonlinear least-squares problem follows Chapter 10 of Dennis and Schnabel [13]. The nonlinear least-squares problem is

$$\text{minimize } f(x) = \tfrac{1}{2} R(x)^t R(x) = \tfrac{1}{2} \sum_{i=1}^{m} r_i(x)^2 \tag{17}$$

where $m \geqq n$, the residual function $R: \mathbf{R}^n \to \mathbf{R}^m$ is nonlinear, and $r_i(x)$ denotes the ith component function of $R(x)$. Straightforward calculations show that the gradient of f is given by

$$\nabla f(x) = J(x)^t R(x)$$

where $J(x)$ denotes the Jacobian of R at x, and the Hessian of f is given by

$$\nabla^2 f(x) = C(x) + S(x)$$

where

$$C(x) = J(x)^t J(x), \qquad S(x) = \sum_{i=1}^{m} r_i(x)\nabla^2 r_i(x),$$

and $\nabla^2 r_i(x)$ is the Hessian of r_i at x.

By a partially known secant method for the nonlinear least-squares problem (17), we mean the iterative procedure defined by (4) and (7) where s is the quasi-Newton step defined by

$$Bs = -\nabla f(x)$$

and y and $y^{\#}$ are approximations to $\nabla^2 f(x_*)s$ and $S(x_*)s$, respectively.

The choice for $y^{\#}$

$$y^{\#} = [J(x_+) - J(x)]^t R(x_+) \tag{18}$$

was suggested independently by Dennis [8] and Bartholomew-Biggs [3] and is currently used in the algorithms given by Dennis, Gay, and Welsch [10] and Al-Baali and Fletcher [1]. Initially, Dennis, Gay, and Welsch [10] used, in the NL2SOL code,

$$y = \nabla f(x_+) - \nabla f(x), \tag{19}$$

and, in the update function, $U^2(x, A)$. It was Al-Baali and Fletcher [1] who first suggested using

$$y = y^{\#} + J(x_+)^t J(x_+)s$$

instead of (19), introducing, in this way, the known part of the problem into the update function. This modification improved the numerical performance of the NL2SOL code [9].

Consider the following standard assumptions for problem (17).

(SA1) Problem (17) has a strong local minimizer x_* with $\nabla^2 f(x_*)$ positive definite.

(SA2) The function $f \in C^2$, and J and $\nabla^2 f$ are locally Hölder continuous at x_*, i.e., there exist $L_1 \geqq 0$, $L_2 \geqq 0$, $\varepsilon \geqq 0$ and $p \in]0, 1]$ such that

$$|J(x) - J(x_*)| \leqq L_1 |x - x_*|^p$$

and

$$|\nabla^2 f(x) - \nabla^2 f(x_*)| \leqq L_2 |x - x_*|^p$$

for $x \in D = \{x \,|\, |x - x_*| < \varepsilon\}$.

The following lemma will serve as the foundation of our convergence result for the nonlinear least-squares algorithm (17) as it was in Dennis, Martínez, and Tapia [11].

LEMMA 7. *Assume that the standard assumptions for problem* (17) *hold. Then, there exists a positive constant K such that*

$$|y^{\#} - S(x_*)s| \leqq K\sigma(x, x_+)|s|$$

where $y^{\#}$ *is given by* (18), $x, x_+ \in D$, *and* $s = x_+ - x$.

The following result is a generalization of Theorem 4.1 of [11].

THEOREM 8. *Assume that the standard assumptions for problem* (17) *hold. Then, there exist positive constants* ε, δ *such that, for* $x_0 \in \mathbf{R}^n$ *and symmetric* $A_0 \in \mathbf{R}^{n \times n}$ *satisfying* $|x_0 - x_*| < \varepsilon$ *and* $|A_0 - S(x_*)| < \delta$, *the iteration sequence* $\{x_k\}$ *generated by the partially known secant method which uses the update function* $U^2(x, A)$ *given by* (7) *for problem* (17) *is q-superlinearly convergent to* x_*.

Proof. The proof of this theorem is a straightforward application of Theorem 6 and Lemma 7. □

4.2. Constrained optimization. We will consider the special case of the nonlinear programming problem where we only have equality constraints, namely,

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g(x) = 0 \end{aligned} \tag{20}$$

where $f: \mathbf{R}^n \to \mathbf{R}$, and $g: \mathbf{R}^n \to \mathbf{R}^m$ are smooth functions ($m \leqq n$).

Associated with problem (20) is the Lagrangian function

$$l(x, \lambda) = f(x) + g(x)^t \lambda.$$

Straightforward calculations show that the gradient of l with respect to x is given by

$$\nabla_x l(x, \lambda) = \nabla f(x) + \nabla g(x)\lambda,$$

and the Hessian of l with respect to x is given by

$$\nabla_x^2 l(x, \lambda) = \nabla^2 f(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 g_i(x),$$

where $g_i : \mathbf{R}^n \to \mathbf{R}$ denotes the ith component function of g.

Following Tapia [22], by the SQP augmented scale secant method for the constrained optimization problem (20), we mean the iterative process

$$x_+ = x + s, \qquad \lambda_+ = \lambda + \Delta\lambda, \qquad A_+ \in U(x, A) \tag{21}$$

where A is a symmetric approximation to $\nabla_x^2 l(x, \lambda)$, and s and $\Delta\lambda$ are, respectively, the solution and the multiplier associated with the solution of the quadratic programming problem

$$\begin{aligned} &\text{minimize} && \nabla_x l(x, \lambda)^t s + \tfrac{1}{2} s^t A s \\ &\text{subject to} && \nabla g(x)^t s + g(x) = 0. \end{aligned} \tag{22}$$

In (22), A is a symmetric approximation to $\nabla_x^2 l(x, \lambda)$, and $U(x, A)$ is the partially known update function given by (7) where

$$\begin{aligned} y^{\#} &= \nabla_x l(x_+, \lambda_+) - \nabla_x l(x, \lambda_+), \\ y &= y^{\#} + \rho \nabla g(x_+) \nabla g(x_+)^t s, \\ B &= A + \rho \nabla g(x_+) \nabla g(x_+)^t \end{aligned} \tag{23}$$

and ρ is the penalty constant in the augmented Lagrangian function

$$L(x, \lambda; \rho) = l(x, \lambda) + \tfrac{1}{2} \rho g^t g, \qquad \rho \geqq 0.$$

Observe that B is a partially known approximation to the Hessian of the augmented Lagrangian at the solution, i.e.,

$$B \approx \nabla_x^2 L(x_*, \lambda_*; \rho) = \nabla_x^2 l(x_*, \lambda_*) + \rho \nabla g(x_*) \nabla g(x_*)^t$$

since the last term of

$$\nabla_x^2 L(x, \lambda; \rho) = \nabla_x^2 l(x, \lambda) + \rho \nabla g(x) \nabla g(x)^t + \rho \sum_{i=1}^{m} g_i(x) \nabla^2 g_i(x)$$

vanishes at the solution x_*. Moreover, Tapia [22] gave strong arguments for ignoring this second-order term in any type of SQP augmented Lagrangian quasi-Newton method.

Three issues are important in the derivation of the SQP augmented scale secant method: first, the use of the augmented Lagrangian instead of the standard Lagrangian to compensate the lack of positive definiteness of $\nabla_x^2 l(x_*, \lambda_*)$; second, the use of the structure of $\nabla_x^2 L(x_*, \lambda_*; \rho)$ as much as possible; and third, the fact that the penalty constant cancels out in all parts of the algorithm except in the *scale* of the secant update (see [22] or [20] for the definition of *scale*).

In fact, the SQP augmented scale secant method is an SQP (standard) Lagrangian secant method with a modified (or augmented) scale (see [20] and [22] for more details). It is this change of scale which takes care of the lack of positive definiteness in the Hessian of the Lagrangian and allows us to use positive-definite secant updates, like the ones from the convex class, for constrained optimization problem (20) without assuming that $\nabla_x^2 l(x_*, \lambda_*)$ is positive definite.

Clearly, since $y^{\#'}s$ is not necessarily positive, the augmented scale secant updates in (22) do not have the hereditary positive-definiteness property. However, they do possess this property on $N(x_+)$ where

$$N(x)=\{z\in\mathbf{R}^n\colon \nabla g(x)^t z=0\} \tag{24}$$

(Tapia [22, Prop. 4.4]).

The following are standard assumptions in the theory of quasi-Newton methods for problem (20).

(SA1) Problem (20) has a local solution x_* with associated multiplier λ_*.

(SA2) The functions f and g_i, $i=1,\cdots,m$ have second derivatives which are locally Hölder continuous at x_*, i.e., there exist $L\geqq 0$, $L_i\geqq 0$, $i=1,\cdots,m$, $\varepsilon\geqq 0$, and $p\in\,]0,1]$ such that

$$|\nabla^2 f(x)-\nabla^2 f(x_*)|\leqq L|x-x_*|^p \tag{25a}$$

and

$$|\nabla^2 g_i(x)-\nabla^2 g_i(x_*)|\leqq L_i|x-x_*|^p \quad i=1,\cdots,m \tag{25b}$$

for $x\in D=\{x\,|\,|x-x_*|<\varepsilon\}$.

(SA3) The matrix

$$\nabla^2 l(x_*,\lambda_*)=\begin{pmatrix}\nabla_x^2 l(x_*,\lambda_*) & \nabla g(x_*)\\ \nabla g(x_*)^t & 0\end{pmatrix}$$

is nonsingular.

To develop the local convergence theory for problem (20), we will use the following well-known results of Avriel ([2, Cor. 12.9. Thm. 12.10]).

Result 9. Assume (SA1) holds. Then (SA3) is equivalent to the following two statements:

(SA3a) The matrix $\nabla g(x_*)$ has full rank.

(SA3b) The matrix $\nabla_x^2 l(x_*,\lambda_*)$ is positive definite on the subspace $N(x_*)$, where $N(x)$ is given by (24).

Result 10. Assume that the standard assumptions for problem (20) hold. Then there exists ρ_* such that $\nabla_x^2 L(x_*,\lambda_*;\rho)$ is positive definite for any $\rho>\rho_*$.

Tapia [22] used the Fontecilla–Steihaug–Tapia [18] and Broyden–Dennis–Moré [5] theories to prove that, under the standard assumptions, the SQP augmented scale BFGS and DFP secant methods were locally and q-superlinearly convergent to x_*. In this section, we will use a similar approach to generalize this result to the SQP augmented scale secant method which uses the update function $U(x,A)$ given by (7). The main difference in our approach is the unified way in which we obtain the bounded deterioration inequality for all the augmented scale secant updates from the convex class. Indeed, this inequality follows from Theorem 4 and the following lemma.

LEMMA 11. *Assume that the standard assumptions for problem* (20) *hold. Then, there exists a positive constant K such that*

$$|y^{\#}-\nabla_x^2 l(x_*,\lambda_*)s|\leqq K\sigma(x,x_+)|s| \tag{26}$$

where $y^{\#}$ *is given by* (23), $x,\ x_+\in D$, *and* $s=x_+-x$.

Proof. Observe that by adding and subtracting the appropriate term we have

$$\begin{aligned} y^{\#} - \nabla_x^2 l(x_*, \lambda_*)s &= \nabla_x l(x_+, \lambda_+) - \nabla_x l(x, \lambda_+) - \nabla_x^2 l(x_*, \lambda_*)s \\ &= \nabla f(x_+) + \nabla g(x_+)\lambda_+ - \nabla f(x) - \nabla g(x)\lambda_+ \\ &\quad - \nabla^2 f(x_*)s - \sum_{i=1}^{m} \lambda_*^i \nabla^2 g_i(x_*)s \\ &= \nabla f(x_+) - \nabla f(x) - \nabla^2 f(x_*)s \\ &\quad + \sum_{i=1}^{m} [\nabla g_i(x_+) - \nabla g_i(x) - \nabla^2 g_i(x_*)s]\lambda_*^i \\ &\quad + \sum_{i=1}^{m} [\nabla g_i(x_+) - \nabla g_i(x) - \nabla^2 g_i(x_*)s][\lambda_+^i - \lambda_*^i] \\ &\quad + \sum_{i=1}^{m} [\lambda_+^i - \lambda_*^i]\nabla^2 g_i(x_*)s \end{aligned} \tag{27}$$

where λ_+^i and λ_*^i are the ith component of λ_+ and λ_*, respectively.

From (25) and Lemma 4.1.15 of Dennis and Schnabel [13] we have

$$|\nabla f(x_+) - \nabla f(x) - \nabla^2 f(x_*)s| \leqq L\sigma(x, x_+)|s| \tag{28a}$$

and

$$|\nabla g_i(x_+) - \nabla g_i(x) - \nabla^2 g_i(x_*)s| \leqq L_i\sigma(x, x_+)|s|, \quad i = 1, \cdots, m. \tag{28b}$$

Therefore, using (27), (28), and the Cauchy-Schwarz inequality

$$\begin{aligned} |y^{\#} - \nabla_x^2 l(x_*, \lambda_*)s| &\leqq L\sigma(x, x_+)|s| + \sum_{i=1}^{m} L_i|\lambda_*^i|\sigma(x, x_+)|s| \\ &\quad + \sum_{i=1}^{m} L_i|\lambda_+^i - \lambda_*^i|\sigma(x, x_+)|s| + \sum_{i=1}^{m} \bar{L}_i|\lambda_+^i - \lambda_*^i||s| \\ &\leqq \left[L + \sum_{i=1}^{m} L_i|\lambda_*^i|\right]\sigma(x, x_+)|s| \\ &\quad + \left[\left(\sum_{i=1}^{m} L_i^2\right)^{1/2}\varepsilon + \left(\sum_{i=1}^{m} \bar{L}_i^2\right)^{1/2}\right]|\lambda_+ - \lambda_*||s| \end{aligned} \tag{29}$$

where $\bar{L}_i = |\nabla^2 g_i(x_*)|$.

From Proposition 4.2 of Fontecilla, Steihaug, and Tapia [18] we have that there exists a positive constant γ such that

$$|\lambda_+ - \lambda_*| \leqq \gamma|x - x_*| \tag{30}$$

for all x close enough to x_*.

Therefore, using (29) and (30), we establish (26) with

$$K = L + \sum_{i=1}^{m} L_i|\lambda_*^i| + \gamma\left[\left(\sum_{i=1}^{m} L_i^2\right)^{1/2}\varepsilon + \left(\sum_{i=1}^{m} \bar{L}_i^2\right)^{1/2}\right]. \tag{31}$$

□

THEOREM 12. *Assume that the standard assumptions for problem* (20) *hold and* $\rho \geqq 0$ *has been chosen so that* $\nabla_x^2 l(x_*, \lambda_*)$ *is positive definite (see Result* (10)). *Then, there exist positive constants* ε, δ *such that, for* $x_0 \in \mathbf{R}^n$ *and symmetric* $A_0 \in \mathbf{R}^n$ *satisfying* $|x_0 - x_*| < \varepsilon$ *and* $|A_0 - \nabla_x^2 l(x_*, \lambda_*)| < \delta$, *the iteration sequence* $\{x_k\}$ *generated by the* SQP *augmented scale secant method which uses the update function* $U^2(x, A)$ *given* (7) *is q-superlinearly convergent to* x_*.

Proof. First, let us remember that the quadratic problem (22) would have the same solution if we used B and $\nabla_x L(x, \lambda; \rho)$ instead of A and $\nabla_x l(x, \lambda)$, respectively (Tapia [22, Prop. 3.1]). Now, the bounded deterioration inequality for B, the partially known secant approximation to $\nabla_x^2 L(x_*, \lambda_*; \rho)$, follows from Lemma 11 and Theorem 4 for the augmented scale secant update function $U^2(x, A)$ given by (7). In turn, this bounded deterioration inequality allows us to use Theorem 3.1 of Fontecilla, Steihaug, and Tapia [18] to establish the existence of the constants ε, δ and the q-linear convergence of the sequence $\{x_k\}$. Then, using an argument identical to the one we used in Theorem 6, we can prove

$$\lim_k \frac{|[B_k - \nabla_x^2 L(x_*, \lambda_*; \rho)]s_k|}{|s_k|} = 0. \tag{32}$$

Finally, the q-superlinear convergence follows from Corollary 5.4 of Fontecilla, Steihaug, and Tapia [18]. □

5. Conclusions. In this paper we have proved local and superlinear convergence for partially known quasi-Newton updates generated from the convex Broyden class. This theory allowed us to generalize both the results given by Dennis, Martínez, and Tapia [11] and those given by Tapia for equality-constrained optimization problems [22]. We feel that these updates are of great potential value as it has been shown by the popular NL2SOL code [10]. There is a very good chance that, by imposing some restrictions on the computed part $C(x)$ of the Hessian, the results of this paper could be extended to obtain global convergence theorems of the sort given by Powell [21] and Byrd, Nocedal, and Yuan [6].

Acknowledgments. Special thanks go to Richard A. Tapia and John E. Dennis, Jr. Their wise advice, opportune suggestions, and encouragement made this work really easy.

REFERENCES

[1] M. Al-Baali and R. Fletcher, *Variational methods for non-linear least-squares*, J. Oper. Res. Soc., 36 (1985), pp. 405–421.

[2] M. Avriel, *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[3] M. C. Bartholomew-Biggs, *The estimation of the Hessian matrix in nonlinear least squares problems with non-zero residuals*, Math. Programming, 12 (1977), pp. 67–80.

[4] C. G. Broyden, *Quasi-Newton methods and their application to function minimization*, Math. Comp., 21 (1967), pp. 368–381.

[5] C. G. Broyden, J. E. Dennis, Jr., and J. J. Moré, *On the local and superlinear convergence of quasi-Newton methods*, J. Inst. Math. Appl., 12 (1973), pp. 223–245.

[6] R. H. Byrd, J. Nocedal, and Y. Yuan, *Global convergence of a class of quasi-Newton methods on convex problems*, SIAM J. Numer. Anal., 24 (1987), pp. 1152–1170.

[7] J. E. Dennis, Jr., *Toward a unified convergence theory for Newton-like method*, in Nonlinear Functional Analysis and Applications, L. B. Rall, ed., Academic Press, New York, 1971.

[8] ———, *A brief survey of convergence results for quasi-Newton methods*, in Nonlinear Programming, SIAM-AMS Proceedings, R. Cottle and C. Lemke, eds., 1976.

[9] ———, Private communication, Department of Mathematical Sciences, Rice University, Houston, TX, April 1987.

[10] J. E. Dennis, Jr., D. M. Gay, and R. E. Welsch, *An adaptative nonlinear least-squares algorithm*, ACM Trans. Math. Software, 7 (1981), pp. 348–368.

[11] J. E. Dennis, Jr., H. J. Martínez, and R. A. Tapia, *Convergence theory for structured* BFGS *secant method with an application to nonlinear least squares*, J. Optim. Theory Appl., 61 (1989), pp. 161–178.

[12] J. E. Dennis, Jr. and J. J. Moré, *A characterization of superlinear convergence and its application to quasi-Newton methods*, Math. Comp., 28 (1974), pp. 549–560.

[13] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[14] J. E. DENNIS, JR. AND H. F. WALKER, *Convergence theorems for least-change secant updates methods*, SIAM J. Numer. Anal., 18 (1981), pp. 949-987.

[15] J. R. ENGELS, *Simulation and optimization of macro-econometric models: a new computational method*, SIAM Conference on Optimization, Houston, TX, May 18-20, 1987.

[16] ———, *Local convergence analysis for partially known quasi-Newton updates*, Tech. Report 88/13, Department of Mathematics, Facultés Universitaires ND de la Paix, B-5000 Namur, Belgium, 1988.

[17] R. FLETCHER, *A new approach to variable metric algorithms*, Comput. J., 13 (1970), pp. 317-322.

[18] R. FONTECILLA, T. STEIHAUG, AND R. TAPIA, *A convergence theory for a class of quasi-Newton methods for constrained optimization*, SIAM J. Numer. Anal., 24 (1987), pp. 1133-1152.

[19] A. GRIEWANK AND P. L. TOINT, *Local convergence analysis for partitioned quasi-Newton updates*, Numer. Math., 39 (1982), pp. 429-448.

[20] H. J. MARTÍNEZ, *Local and superlinear convergence of structured secant methods from the convex class*, Ph.D. thesis, Department of Mathematical Sciences, Rice University, Houston, TX, 1988.

[21] M. J. D. POWELL, *Convergence properties of a class of minimization algorithms*, in Nonlinear Programming 2, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, 1975, pp. 1-27.

[22] R. A. TAPIA, *On secant updates for use in general constrained optimization*, Math. Comp., 51 (1988), pp. 181-202.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 57–82, February 1991

006

MINIMIZATION OF LOCALLY LIPSCHITZIAN FUNCTIONS*

JONG-SHI PANG†‡, SHIH-PING HAN†, AND NARAYAN RANGARAJ†

Abstract. This paper presents a globally convergent model algorithm for the minimization of a locally Lipschitzian function. The algorithm is built on an iteration function of two arguments, and the convergence theory is developed parallel to analogous results for the problem of solving systems of locally Lipschitzian equations. Application of the theory to a wide range of nonsmooth optimization problems is discussed. These include the minimax problem, the composite optimization problem, the implicit programming problem, and others. A recently developed nonmonotone linesearch technique is shown to be applicable in this nonsmooth context, and an extension to constrained problems is also presented.

Key words. nonsmooth optimization, locally Lipschitzian functions, Dini stationary points, global convergence, minimax problem, implicit programming

AMS(MOS) subject classifications. 90C30, 90C33

1. Introduction.

Starting with the seminal work of Clarke [4], there has been a growing amount of research in the area of nonsmooth optimization. Broadly speaking, this topic is concerned with the optimization of an objective function which is not differentiable in the traditional sense of Fréchet or Gâteau; constraints defined by nonsmooth functions may also be present.

Much of the algorithmic development in nonsmooth optimization has been based on the notion of the subgradient as well as on Clarke's generalized subdifferential or their variants. There are several texts which provide a good summary of various solution methods in this area [6], [10], [17], [35], as well as a large body of published articles which include [2], [9], [14], [18], [19], [20], [22], [23], [36], to name just a few.

In the present paper, we propose a fairly general model algorithm for minimizing a nonsmooth function which is assumed only to be locally Lipschitz continuous. We develop a convergence theory for the algorithm and discuss the specialization of the theory to some specific classes of nondifferentiable optimization problems. Extensions of the algorithm to allow for nonmonotone linesearch and to treat problems with constraints will also be discussed.

The main difference between our model algorithm and many existing methods for nonsmooth optimization problems is that we make no explicit use of the concept of subgradients; instead, it is built on an abstract function of two variables which is used to define the direction-finding subproblem at each iteration. Under appropriate assumptions, our algorithm will compute what we call a *Dini stationary point* of a locally Lipschitzian function. The development in the present paper is closely related to that in the recent paper [15] in which a similar theory is established for the problem of solving systems of nonsmooth equations by Newton-type descent methods.

The research of this paper is partly motivated by our interest in developing some robust numerical methods for solving the class of mathematical programming problems subject to equilibrium constraints [16]. Such problems have diverse applications and include the so-called bilevel programming problem [1], the continuous network design

* Received by the editors April 16, 1990; accepted for publication (in revised form) June 13, 1990.

† Department of Mathematical Sciences, The Whiting School of Engineering, The Johns Hopkins University, Baltimore, Maryland 21218.

‡ The work of this author was based on research supported by National Science Foundation grant ECS-8717968.

problem in transportation systems [21], and the Stackelberg game problem in oligopolistic models of competition [33]. In general, these problems are very difficult to solve due to their intrinsic nonconvexity and nondifferentiability; they are not easily amenable to solution by existing methods for nondifferentiable problems. There have been many heuristics proposed under very restrictive smoothness/differentiability assumptions [7], [11]. We shall consider an important subclass of these mathematical programs with equilibrium constraints and discuss how the specialization of our model algorithm results in an effective descent method for solving these problems. The convergence of the resulting method requires only a single differentiability assumption at the candidate solution; this is a significant improvement over the existing methods, which essentially treat the nonsmooth problem as a smooth problem by assuming the continuous differentiability condition throughout.

The organization of this paper is as follows. In the next section, we define the concept of a Dini stationary point of a locally Lipschitzian function, describe the model algorithm, and establish its convergence. Section 3 demonstrates how various known algorithms for special classes of nonsmooth optimization problems can be fitted into this framework. In § 4, we discuss the *implicit programming problem*; this is the special case of the mathematical programming problem with equilibrium constraints that we mentioned above. Finally, the last two sections extend the general theory to allow for nonmonotone linesearches and to handle constrained problems.

2. A model descent method. Let $\theta: R^n \to R$ be a locally Lipschitzian function. Consider the optimization problem

$$\text{minimize} \quad \theta(x): x \in R^n. \tag{1}$$

In this section, we describe a globally convergent descent method for solving the problem (1). Before doing this, we review some concepts involving directional derivatives and define the notion of a Dini stationary point of the function θ. Most of the derivative ideas reviewed have been widely used in the nonsmooth optimization literature (see [4], [17]).

2.1. Directional derivatives and Dini stationary points. We first recall some familiar definitions. For a function $\theta: R^n \to R$, the (usual) *directional derivative* of θ at $x \in R^n$ in the direction $d \in R^n$ is defined to be

$$\theta'(x, d) := \lim_{\lambda \to 0^+} \frac{\theta(x+\lambda d) - \theta(x)}{\lambda}$$

if the limit exists. The function θ is said to be *directionally differentiable* at x if $\theta'(x, d)$ exists for all $d \in R^n$. A slightly more general concept is the *upper Dini directional derivative* of θ at x in the direction d; this is defined to be

$$\theta^D(x, d) := \limsup_{\lambda \to 0^+} \frac{\theta(x+\lambda d) - \theta(x)}{\lambda}.$$

The *Clarke generalized directional derivative* of θ at x in the direction d is defined to be

$$\theta^0(x, d) := \limsup_{\lambda \to 0^+,\, y \to x} \frac{\theta(y+\lambda d) - \theta(y)}{\lambda}.$$

In general, both the upper Dini directional derivative $\theta^D(x, d)$ and the Clarke directional derivative $\theta^0(x, d)$ are well defined and finite if θ is locally Lipschitzian at

x; whereas the usual directional derivative $\theta'(x, d)$ need not exist for such a θ. Note that if θ is locally Lipschitzian at x, then,

$$\theta^D(x, d) = \limsup_{v \to d,\, \lambda \to 0^+} \frac{\theta(x+\lambda v) - \theta(x)}{\lambda},$$

and a similar expression holds for the Clarke derivative $\theta^0(x, d)$. Clearly, we have

$$\theta^0(x, d) \geqq \theta^D(x, d), \tag{2}$$

and $\theta^D(x, d) = \theta'(x, d)$ for all $d \in R^n$ if θ is directionally differentiable at x. Moreover, all three directional derivatives are positively homogeneous in d for each fixed x; i.e., for all scalars $\lambda \geqq 0$,

$$\theta^0(x, \lambda d) = \lambda\theta^0(x, d), \qquad \theta^D(x, \lambda d) = \lambda\theta^D(x, d) \quad \text{and} \quad \theta'(x, \lambda d) = \lambda\theta'(x, d).$$

It is not difficult to show that if θ is locally Lipschitzian at x, then both the upper Dini directional derivative $\theta^D(x, d)$ and the Clarke directional derivative $\theta^0(x, d)$ are Lipschitz continuous functions in d with the same modulus as θ [4]. The same conclusion holds for the directional derivative $\theta'(x, d)$ if θ is directionally differentiable at x.

The function θ is said to be *subdifferentiably regular* at x if θ is directionally differentiable at x and $\theta^0(x, d) = \theta'(x, d)$ for all d [4]. The function θ is said to be *strongly F(réchet)-differentiable* or to have a *strong F-derivative* at x if the gradient vector $\nabla\theta(x)$ exists and the following limit condition holds:

$$\lim_{(u, v) \to (x, x)} \frac{\theta(u) - \theta(v) - \nabla\theta(x)^T(u - v)}{\|u - v\|} = 0.$$

It is easy to see that if θ has a strong F-derivative at x, then θ is subdifferentiably regular there; moreover, we have $\theta^0(x, d) = \theta^D(x, d) = \theta'(x, d) = \nabla(x)^T d$ for all $d \in R^n$. It is well known that a convex function $\theta: R^n \to R$ is subdifferentiably regular at every point $x \in R^n$.

By applying essentially the same proof of [25, Thm. 2] and replacing the usual directional derivative with the upper Dini directional derivative, we can establish the following result, which relates the strong F-differentiability of θ at a given point x to the continuity of the upper Dini directional derivative $\theta^D(\cdot, d)$ at x.

PROPOSITION 1. *Suppose that $\theta: R^n \to R$ is Lipschitz continuous in a neighborhood of a vector x. Then, the following three statements are equivalent:*

(a) *The upper Dini directional derivative $\theta^D(x, \cdot)$ satisfies the stronger limit property:*

$$\lim_{(u, v) \to (x, x)} \frac{\theta(u) - \theta(v) - \theta^D(x, u - v)}{\|u - v\|} = 0;$$

(b) *θ has a strong F-derivative at x;*

(c) *The upper Dini directional derivative $\theta^D(\cdot, d)$ is continuous at x, and the continuity is "uniform" for each $d \in R^n$, i.e., for each $\varepsilon > 0$, there exists a neighborhood N of x such that for all vectors $y \in N$ and all $d \in R^n$,*

$$|\theta^D(x, d) - \theta^D(y, d)| \leqq \varepsilon\|d\|.$$

In the context of optimization, the directional derivative, if it exists, can be used to describe optimality as follows. If x is a local minimum point of the problem (1), then x is a *stationary point* of θ, i.e.,

$$\theta'(x, d) \geqq 0 \quad \text{for all } d \in R^n. \tag{3}$$

Conversely, if θ is convex, then a vector x satisfying (3) must be a globally optimal solution of (1).

If θ is locally Lipschitzian, we call a vector $x \in R^n$ a *Dini stationary point* of θ if

$$\theta^D(x, d) \geqq 0 \quad \text{for all } d \in R^n. \tag{4}$$

It is easy to see that if x is a local minimum point of a locally Lipschitzian function θ, then x must be a Dini stationary point of θ. If θ is directionally differentiable at x, then x is a Dini stationary point of θ if and only if x is a stationary point of θ. Nevertheless, if θ fails to be directionally differentiable at x but is locally Lipschitzian there, then the concept of x being a (usual) stationary point is meaningless, while that of x being a Dini stationary point remains well defined.

In view of the inequality (2), it follows that if x is a Dini stationary point of θ, then x must be a stationary point of θ in the sense of Clarke [17], i.e., $0 \in \partial\theta(x)$, where $\partial\theta(x)$ is Clarke's generalized gradient:

$$\partial\theta(x) = \{a \in R^n: \theta^0(x, d) \geqq a^T d \text{ for all } d \in R^n\}.$$

Nevertheless, if θ is not subdifferentiably regular at x, then it is possible for x to be a point such that $0 \in \partial\theta(x)$, but x is not a Dini stationary point of θ. For example, if

$$\theta(x) = -|x|, \qquad x \in R,$$

then $\theta^0(0, d) = |d|$ for all $d \in R$; thus, $x = 0$ satisfies the property $0 \in \partial\theta(x)$. Since $\theta^D(0, d) = \theta'(0, d) = -|d|$, clearly $x = 0$ is not a Dini stationary point of this function θ.

In its most general form, the model algorithm to be described later is designed to compute a Dini stationary point of a locally Lipschitzian function θ.

2.2. Background of method. Like many descent methods for minimizing a smooth objective function, our method for solving the problem (1) is iterative; each iteration consists of two major steps: a direction-finding step and a linesearch step. In traditional smooth optimization, the direction-finding step typically involves the minimization of a quadratic approximation of θ which depends on the gradient vector of θ at the current iterate. More specifically, if x^k is given, we compute a search direction d^k by solving the problem

$$\text{minimize} \quad \nabla\theta(x^k)^T d + \tfrac{1}{2} d^T B_k d \tag{5}$$

where B_k is a symmetric positive-definite matrix which is, presumably, an approximation of the Hessian matrix $\nabla^2\theta(x^k)$. The linesearch step involves computing a stepsize $\tau_k > 0$ so that the next iterate $x^{k+1} = x^k + \tau_k d^k$ satisfies a certain rule which yields "sufficient decrease" in the objective function θ.

When the gradient vector $\nabla\theta(x^k)$ does not exist, but θ is locally Lipschitzian, we could, in principle, replace the direction-finding problem (5) by the following one:

$$\text{minimize} \quad \theta^D(x^k, d) + \tfrac{1}{2} d^T B_k d. \tag{6}$$

Nevertheless, as stated in Proposition 1, if θ is not Fréchet differentiable, the directional derivative $\theta^D(x, d)$ need not be continuous in x for each fixed d. As we shall see, such continuity of the Dini directional derivative is essential for the convergence of a descent method that makes use of problem (6) to generate the search direction at each iteration.

In summary, the nondifferentiability of θ has created two technical issues when either one of the subproblems, (5) or (6), is used as a direction-finding procedure. One is the nonexistence of the gradient vector $\nabla\theta(x)$; the other is the discontinuity of the

directional derivative $\theta^D(x, d)$ in the variable x. The algorithm to be described next is a proposal to circumvent these two difficulties.

2.3. Description of method. Consider the problem (1) where $\theta: R^n \to R$ is locally Lipschitzian. Let x^k be a given iterate. We consider the following subproblem to obtain a search direction d^k:

$$\text{minimize} \quad \psi(x^k, d) + \tfrac{1}{2} d^T B_k d: \ d \in R^n \tag{7}$$

where $\psi: R^n \times R^n \to R$ is a given function and $B_k \in R^{n \times n}$ is a given symmetric matrix. Examples of the function ψ corresponding to specific functions θ will be given in § 3. For now, we make the following blanket assumptions on the function ψ and the sequence of matrices $\{B_k\}$:

(A1) For each fixed vector x, the function $\psi(x, d)$ is continuous in the variable d, and $\psi(x, 0) = 0$. Moreover, for all $(x, d) \in R^n \times R^n$,

$$\psi(x, d) \geqq \theta^D(x, d); \tag{8}$$

(B) There exist constants $\alpha \geqq \beta > 0$ such that for all $x \in R^n$,

$$\beta x^T x \leqq x^T B_k x \leqq \alpha x^T x \quad \text{for all } k.$$

We do not assume the continuity of the function $\psi(x, d)$ in the x-variable. This is consistent with the previous discussion about the discontinuity of the upper Dini directional derivative in the same variable. The condition (8) says that the iteration function ψ *majorizes* the upper Dini directional derivative θ^D. Another noteworthy point is that unlike the various directional derivatives, the function $\psi(x, d)$ is not assumed to be positively homogeneous or Lipschitz continuous in d.

Besides having the theoretical properties (A1) and others which will be stated later, the iteration function $\psi(x, d)$ should render the subproblem (7) computationally easier to solve than the original problem (1). For the special classes of application problems discussed later, the function $\psi(x, d)$ is a kind of "linearization" of the objective function $\theta(x)$.

Under the two assumptions (A1) and (B), the subproblem (7) always has a globally optimal solution; moreover, any such solution that is nonzero yields a "descent direction" for the function θ at the iterate x^k. More specifically, we can establish the following result.

LEMMA 1. *Let $\theta: R^n \to R$ be locally Lipschitzian. Suppose that assumptions* (A1) *and* (B) *hold.*

(a) *The problem* (7) *has a globally optimal solution, and the optimum objective value is nonpositive.*

(b) *If d^k is any nonzero optimal solution of* (7), *then for any $\sigma \in (0, 1)$, there exists a scalar $\bar{\tau} > 0$ such that for all $\tau \in [0, \bar{\tau}]$,*

$$\theta(x^k + \tau d^k) - \theta(x^k) \leqq -\frac{\sigma}{2} \tau (d^k)^T B_k d^k.$$

Proof. Let $L > 0$ be the Lipschitz modulus of θ at the vector x^k. Then, we have

$$|\theta^D(x^k, d)| \leqq L \|d\|$$

for all vectors $d \in R^n$. By the inequality (8) and the positive definiteness of the matrix B_k, we obtain

$$\psi(x^k, d) + \frac{1}{2} d^T B_k d \geqq -L\|d\| + \frac{\beta}{2} d^T d.$$

Hence, it follows that

$$\lim_{\|d\|\to\infty} (\psi(x^k, d)+\tfrac{1}{2}d^T B_k d) = \infty.$$

Consequently, the objective function of (7) is coercive. Thus, an optimal solution of (7) exists. Since $\psi(x^k, 0) = 0$, it follows that the optimum objective value of (7) is always nonpositive. This proves part (a).

To prove part (b), let d^k be a nonzero optimal solution of (7). Then,

$$\psi(x^k, d^k) \leqq -\tfrac{1}{2}(d^k)^T B_k d^k. \tag{9}$$

Suppose that no $\bar{\tau} > 0$ as stated exists. Then there exists a sequence $\{\tau_l\}$ of positive scalars converging to zero, such that for each l,

$$\theta(x^k + \tau_l d^k) - \theta(x^k) > -\frac{\sigma}{2}\tau_l (d^k)^T B_k d^k.$$

Dividing both sides by τ_l, passing to the limit $l \to \infty$, and using the definition of the *upper Dini directional derivative* $\theta^D(x^k, d^k)$, *we deduce that*

$$\theta^D(x^k, d^k) \geqq -\frac{\sigma}{2}(d^k)^T B_k d^k.$$

Thus, condition (A1) implies

$$\psi(x^k, d^k) \geqq \theta^D(x^k, d^k) > -\tfrac{1}{2}(d^k)^T B_k d^k$$

where the last inequality holds because $\sigma \in (0, 1)$ and $(d^k)^T B_k d^k$ is positive. But this contradicts (9). □

Remark. Throughout the discussion, we do not require that the subproblem (7) has a unique solution. Note also that the conclusion of part (b) in the above lemma remains valid as long as d^k is a nonzero direction satisfying the inequality (9).

The above lemma establishes that if the problem (7) has a nonzero optimal solution, then that solution constitutes a "descent direction" of θ at the iterate x^k. We want to establish the converse of this conclusion. In other words, we wish to be able to show that if the zero vector is the only globally optimal solution of (7), then x^k is a Dini stationary point of θ. The next result gives a sufficient condition for this converse statement to hold.

PROPOSITION 2. *In addition to the assumptions of Lemma* 1, *suppose the following condition holds*: (A2) *for all vectors* $d \in R^n$,

$$\liminf_{\lambda \to 0^+} \frac{\psi(x^k, \lambda d)}{\lambda} \geqq \theta^D(x^k, d). \tag{10}$$

Then, the following statements are equivalent:

(a) $d = 0$ *is the only globally optimal solution of the problem* (7);

(b) $d = 0$ *is a globally optimal solution of the problem* (7);

(c) *The optimum objective value of the problem* (7) *is zero*;

(d) x^k *is a Dini stationary point of* θ, *i.e.*, $\theta^D(x^k, d) \geqq 0$ *for all* $d \in R^n$.

Proof. (a)⇒(b)⇒(c). These implications are obvious.

(c)⇒(d). Since the optimum objective value of (7) is zero, we have, for all vectors $d \in R^n$ and all scalars $\lambda > 0$,

$$0 \geqq \psi(x^k, \lambda d) + \frac{\lambda^2}{2} d^T B_k d.$$

Dividing by λ, letting $\lambda \downarrow 0$, and invoking assumption (A2), we deduce the desired conclusion.

(d)$\Rightarrow$(a). Suppose that the problem (7) has a nonzero optimal solution d^k. Then the inequality (9) implies $\psi(x^k, d^k) < 0$. By assumption (A1), it follows that $\theta^D(x^k, d^k) < 0$; this contradicts the assumption that x^k is a Dini stationary point of θ. □

Remarks. (1) *Since the upper Dini directional derivative* $\theta^D(x^k, d)$ is positively homogeneous in d, it follows that under the combined assumptions (A1) and (A2), equality must hold in (10).

(2) The only place in the proof of Proposition 2 where assumption (A2) is used is to prove the implication [(c)$\Rightarrow$(d)].

With Lemma 1 and Proposition 2, we may state the following algorithm for computing a Dini stationary point of the problem (1).

The Model Algorithm. Let $\rho, \sigma \in (0, 1)$ be given. Let $x^0 \in R^n$ be arbitrary. Set $k = 0$. In general, given x^k, let d^k be an (arbitrary) globally optimal solution of (7). Terminate if the optimum objective value of (7) is zero; in this case, x^k is a desired Dini stationary point of θ (assuming that the (A1) and (A2) are in force). Otherwise, let m_k be the smallest nonnegative integer m such that

$$\theta(x^k + \rho^m d^k) - \theta(x^k) \leqq -\frac{\sigma}{2} \rho^m (d^k)^T B_k d^k. \tag{11}$$

Set $x^{k+1} = x^k + \rho^{m_k} d^k$; test x^{k+1} for convergence. Repeat the general step with $k+1$ replacing k if x^{k+1} fails the convergence test.

The linesearch step in the algorithm follows the usual Armijo rule; according to Lemma 1(b), the integer m_k can be determined after a finite number of trials starting with $m = 0, 1, 2, \cdots$.

2.4. Convergence of the method. Without loss of generality, we assume that the algorithm generates an infinite sequence of iterates $\{x^k\}$ along with an infinite sequence of nonzero directions $\{d^k\}$. The sequence $\{x^k\}$ satisfies

$$\theta(x^{k+1}) \leqq \theta(x^k) - \frac{\sigma}{2} \tau_k (d^k)^T B_k d^k < \theta(x^k)$$

where $\tau_k = \rho^{m_k}$ is the steplength in the kth iteration. Consequently, the sequence of objective values $\{\theta(x^k)\}$ is strictly decreasing. If the function $\theta(x)$ is bounded below, then the sequence $\{\theta(x^k)\}$ converges and hence $\{\theta(x^{k+1}) - \theta(x^k)\} \to 0$. The stepsize rule (11) therefore implies that

$$\lim_{k \to \infty} \tau_k (d^k)^T B_k d^k = 0. \tag{12}$$

Our goal is to establish that every accumulation point of the sequence $\{x^k\}$, if it exists and satisfies certain properties to be stated, is a Dini stationary point of the function θ. A sufficient condition for such an accumulation point to exist is that the sequence $\{x^k\}$ is bounded, which in turn is true if the level set

$$\{x \in R^n : \theta(x) \leqq \theta(x^0)\} \tag{13}$$

is bounded. Under this boundedness assumption, let $\bar{x}$ be an arbitrary accumulation point of $\{x^k\}$. In the sequel, we impose some assumptions on the function ψ at the limit point $\bar{x}$ in order to establish the Dini stationarity property of θ at $\bar{x}$. To motivate these assumptions, we recall that the reason for using an iteration function $\psi(x, d)$

instead of the upper Dini directional derivative $\theta^D(x, d)$ in defining the direction-finding subproblem was to circumvent the discontinuity problem of the directional derivative in the first argument. In essence, two of the assumptions imposed below, (A3) and (A4), have to do with some kind of continuity of the ψ function in that argument. At present, we do not know how to construct such an iteration function $\psi(x, d)$ to satisfy all these assumptions when θ is a general locally Lipschitzian function; as a matter of fact, the function $\psi(x, d)$ with all these properties may not even exist. Nevertheless, in the next section, we shall discuss a variety of nonsmooth optimization problems for which the desired function $\psi(x, d)$ can be identified.

The following are the assumptions imposed on the function ψ at the limit point $\bar{x}$:

(A2) For every $d \in R^n$,

$$\liminf_{\lambda \to 0^+} \frac{\psi(\bar{x}, \lambda d)}{\lambda} \geqq \theta^D(\bar{x}, d);$$

(A3) For every sequence $\{z^k\}$ converging to $\bar{x}$, every convergent sequence $\{v^k\}$ and every sequence of positive scalars $\{\lambda_k\}$ converging to zero,

$$\lim_{k \to \infty} \psi(z^k, v^k) \geqq \limsup_{k \to \infty} \frac{\theta(z^k + \lambda_k v^k) - \theta(z^k)}{\lambda_k}; \tag{14}$$

whenever the limit in the left-hand side exists;

(A4) There exists a scalar $\varepsilon > 0$ such that for every vector $d \in R^n$ satisfying $\|d\| \leqq \varepsilon$ and every sequence $\{z^k\}$ converging to $\bar{x}$,

$$\liminf_{k \to \infty} \psi(z^k, d) \geqq \psi(\bar{x}, d).$$

We explain the above conditions. Assumption (A2) was required in Proposition 2 in order to obtain the Dini stationarity conclusion on the iterate x^k when the subproblem (7) has a zero optimum objective value. In essence, we need this assumption on the limit point $\bar{x}$ for a similar reason. Assumption (A3) is a strengthening of (A1) at the vector $\bar{x}$; indeed, if $z^k = \bar{x}$ and $v^k = v$ for all k, then the limit expression (14) becomes $\psi(\bar{x}, v) \geqq \theta^D(\bar{x}, v)$ which is exactly the condition (8) at $\bar{x}$. Assumption (A4) is a weak upper semicontinuity property of the function $\psi(\cdot, d)$ at the vector $\bar{x}$ for all vectors d whose norms are sufficiently small; indeed, if "liminf" were replaced by "limsup," then (A4) would become the usual upper semicontinuity property. It is interesting to note that this weak upper semicontinuity property is required to hold only for "small" d and not for all d. We shall give an example later of a function ψ for which (A4) holds but $\psi(\cdot, d)$ is not weakly upper semicontinuous at $\bar{x}$ for all d.

Before stating the main convergence result, we derive a useful consequence of assumption (A1). The following result says that although no continuity assumption in the x variable is imposed on the function $\psi(x, d)$ when d is fixed but arbitrary, (A1) implies that ψ is lower semicontinuous in both arguments at a vector $(x, 0)$ with a zero second argument and an arbitrary first argument.

LEMMA 2. *Let $\theta: R^n \to R$ be locally Lipschitzian. Suppose that the inequality* (8) *holds for all vectors $(x, d) \in R^n \times R^n$. Then, for any sequence $\{(z^k, d^k)\}$ converging to $(x, 0)$ for some $x \in R^n$,*

$$\lim_{k \to \infty} \psi(z^k, d^k) \geqq 0$$

whenever the limit on the left-hand side exists.

Proof. In view of the inequality (8), it suffices to prove

$$\lim_{k\to\infty} \theta^D(z^k, d^k) = 0. \tag{15}$$

Since the sequence $\{z^k\}$ is convergent and the function θ is locally Lipschitzian, it follows that there exists a constant $L>0$ such that for all k,

$$|\theta^D(z^k, d^k)| \leqq L\|d^k\|,$$

which clearly implies the desired limit condition (15). □

We now state and prove the main convergence result.

THEOREM 1. *Let $\theta: R^n \to R$ be a locally Lipschitzian function which is bounded below on R^n. Let $\psi: R^n \times R^n \to R$ and $\{B_k\}$ also be given. Suppose that assumptions* (A1) *and* (B) *hold. Let $\{x^k\}$ be an infinite sequence generated by the model algorithm as described above. If $\bar{x}$ is an accumulation point of $\{x^k\}$ where assumptions* (A2)-(A4) *are satisfied, then $\bar{x}$ is a Dini stationary point of θ.*

Proof. Let $\{x^k: k\in K\}$ be a subsequence of $\{x^k\}$ converging to $\bar{x}$. Let $\{d^k: k\in K\}$ be a corresponding sequence of directions generated by the algorithm. By construction, each direction d^k is an optimal solution of the subproblem (7) whose optimum objective value is negative (because of the infinite nature of the sequence $\{x^k\}$).

We first show that the sequence of directions $\{d^k: k\in K\}$ is bounded. By the optimality of d^k and assumptions (A1) and (B), we have for each $k\in K$,

$$|\theta^D(x^k, d^k)| \geqq -\psi(x^k, d^k) > \frac{1}{2}(d^k)^T B_k d^k \geqq \frac{\beta}{2}\|d^k\|^2. \tag{16}$$

Furthermore, as noted in the proof of Lemma 2, there is a constant $L>0$ such that for all $k\in K$,

$$|\theta^D(x^k, d^k)| \leqq L\|d^k\|;$$

this, along with the previous inequalities, gives

$$\|d^k\| \leqq \frac{2L}{\beta}.$$

The boundedness of the sequence $\{d^k: k\in K\}$ is thus established. This implies, by (16), that the sequence $\{\psi(x^k, d^k): k\in K\}$ is also bounded.

By restricting our discussion to suitable subsequences if necessary, we assume that both the sequence of directions $\{d^k: k\in K\}$ and the sequence of matrices $\{B_k: k\in K\}$ converge, respectively, to some vector $\bar{d}$ and some matrix $\bar{B}$. We may further assume that

$$\lim_{k\to\infty,\, k\in K} \psi(x^k, d^k) \tag{17}$$

exists. Clearly, the limit matrix $\bar{B}$ is positive definite.

Recalling the limit condition (12), we consider two cases, depending on whether the liminf of the sequence of stepsizes $\{\tau_k: k\in K\}$ is positive or zero. Suppose

$$\liminf_{k\to\infty,\, k\in K} \tau_k > 0.$$

Then, condition (12) yields

$$\bar{d}^T\bar{B}\bar{d} = \lim_{k\to\infty,\, k\in K} (d^k)^T B_k d^k = 0,$$

which in turn implies $\bar{d}=0$ because $\bar{B}$ is positive definite.

By the optimality of d^k, we have, for all vectors $d \in R^n$,

$$\psi(x^k, d^k) + \tfrac{1}{2}(d^k)^T B_k d^k \leqq \psi(x^k, d) + \tfrac{1}{2} d^T B_k d.$$

By taking liminf on both sides as $\{k \to \infty, k \in K\}$, the left-hand sum approaches zero by the fact that $\{d^k : k \in K\}$ tends to $\bar{d}$ which has just been proved to equal zero, and also by Lemma 2. Thus, by (A4), we deduce, for all $d \in R^n$ satisfying $\|d\| \leqq \varepsilon$,

$$0 \leqq \psi(\bar{x}, d) + \tfrac{1}{2} d^T \bar{B} d.$$

As in the proof of Proposition 2, it follows that for all $d \in R^n$ with $\|d\| \leqq \varepsilon$,

$$\theta^D(\bar{x}, d) \geqq 0.$$

Since the upper Dini directional derivative is positively homogeneous in d, the last inequality must hold for all vectors $d \in R^n$. Thus, $\bar{x}$ is a Dini stationary point of θ, as desired.

Now consider the other case where the sequence of stepsizes $\{\tau_k : k \in K\}$ becomes arbitrarily small in the limit, i.e., suppose

$$\liminf_{k \to \infty,\, k \in K} \tau_k = 0.$$

Without loss of generality, we may assume that $\lim_{k \to \infty,\, k \in K} \tau_k = 0$. Then, we must have

$$\lim_{k \to \infty,\, k \in K} m_k = \infty.$$

By the definition of m_k, it follows that for each k,

$$\theta(x^k + \tau_k' d^k) - \theta(x^k) > -\frac{\sigma}{2} \tau_k' (d^k)^T B_k d^k \tag{18}$$

where $\tau_k' = \rho^{m_k - 1}$. Note that we also have

$$\lim_{k \to \infty,\, k \in K} \tau_k' = 0.$$

Dividing both sides in the expression (18) by τ_k', passing to the limit $k \to \infty$, $k \in K$, using (A3) and the fact that the limit (17) exists, we deduce

$$\lim_{k \to \infty,\, k \in K} \psi(x^k, d^k) \geqq -\frac{\sigma}{2} \bar{d}^T \bar{B} \bar{d}.$$

On the other hand, the left-hand term of the above inequality is no greater than $-\frac{1}{2} \bar{d}^T \bar{B} \bar{d}$ by taking limits in (9). Consequently, we obtain

$$-\frac{1}{2} \bar{d}^T \bar{B} \bar{d} \geqq -\frac{\sigma}{2} \bar{d}^T \bar{B} \bar{d}.$$

Since $\sigma < 1$, we must have $\bar{d} = 0$. By repeating the argument of the previous case, we deduce that $\bar{x}$ is a Dini stationary point of θ. □

3. Applications. The convergence of the model algorithm has been established in a fairly general context; an important practical issue is the construction of a function $\psi(x, d)$ which satisfies the assumptions (A1)–(A4). Although we do not know how to construct such a function for an arbitrary Lipschitzian function θ, the discussion in this section gives an explicit expression of $\psi(x, d)$ for a number of interesting functions θ that arise from various application areas.

This unified approach to the convergence theory for nonsmooth optimization problems gives a common framework for establishing the global convergence of many existing methods for solving these problems. We hope that by identifying what seem to be the essential requirements in a typical convergence proof—through the use of the $\psi(x, d)$ function of two variables that satisfies the stated properties—we have provided a basis for the design of effective algorithms for other nonsmooth problems not considered here.

It is useful to point out that in several of the application problems discussed herein, the iteration function $\psi(x, d)$ is piecewise linear in d for each fixed x; thus, the direction-finding subproblem (7) is a piecewise quadratic program with the quadratic part coming from the term $d^T B_k d$. Admittedly, the numerical solution of a piecewise quadratic program has not been very well studied within mathematical programming; there are some scattered papers [31], [32], [34]. We believe that this class of mathematical programs plays a central role throughout the subject of nonsmooth optimization and deserves a more careful investigation.

3.1. The use of the upper Dini derivative. We begin by studying the choice

$$\psi(x, d) = \theta^D(x, d) \tag{19}$$

for an arbitrary locally Lipschitzian function θ and state a convergence property for the resulting descent method. With the choice (19), assumptions (A1) and (A2) are trivially satisfied. Note that if the algorithm terminates finitely with the subproblem (7) having a zero optimum objective value, then the current iterate x^k must be a Dini stationary point of θ (see Proposition 2). So, we may assume that the algorithm generates an infinite sequence of iterates $\{x^k\}$. The following consequence of Theorem 1 gives the main convergence result for the model algorithm in which the iteration function ψ is chosen to be the upper Dini directional derivative.

COROLLARY 1. *Let $\theta: R^n \to R$ be a locally Lipschitzian function which is bounded below on R^n. Let $\{B_k\}$ be a sequence of matrices satisfying assumption* (B). *Suppose that $\bar{x}$ is an accumulation point of an infinite sequence $\{x^k\}$ generated by the model algorithm with the choice* (19). *If θ is strongly F-differentiable at $\bar{x}$, then $\nabla\theta(\bar{x}) = 0$; in particular, $\bar{x}$ is a stationary point of θ.*

Proof. It suffices to verify conditions (A3) and (A4). Under the strong F-differentiability assumption on θ at $\bar{x}$, we have $\theta^D(\bar{x}, v) = \nabla\theta(\bar{x})^T v$ for all $v \in R^n$. Moreover, for all sequences $\{z^k\}$, $\{v^k\}$, and $\{\lambda_k\}$, as stated in assumption (A3),

$$\lim_{k\to\infty} \frac{\theta(z^k + \lambda_k v^k) - \theta(z^k) - \lambda_k \theta^D(\bar{x}, v^k)}{\lambda_k} = 0,$$

and by Proposition 1,

$$\lim_{k\to\infty} (\theta^D(z^k, v^k) - \theta^D(\bar{x}, v^k)) = 0.$$

The two required assumptions (A3) and (A4) are now easily satisfied as a result of these two limit conditions. □

In spite of the fact that this result requires a fairly strong differentiability assumption about the function θ at the limit point $\bar{x}$, the result is useful in a situation where there is no obvious way of constructing a function ψ satisfying the assumptions (A1)-(A4). It is interesting to note that if the iteration function $\psi(x, d)$ is chosen to be the Clarke directional derivative $\theta^0(x, d)$, then, in the absence of a strong F-differentiability condition at $\bar{x}$, the only assumption that is possibly violated is (A3).

3.2. Composite convex functions. Our next application is that of a *composite convex minimization problem.* For this problem, we exhibit an iteration function ψ for which the satisfaction of assumptions (A3) and (A4) requires no strong F-differentiability property at $\bar{x}$.

Let $h: R^m \to R$ be a convex function and $f: R^n \to R^m$ be a continuously differentiable function. Note that h is not assumed differentiable and the component functions of f are not assumed convex. The composite function $\theta: R^n \to R$

$$\theta(x) := (h \circ f)(x) = h(f(x)) \tag{20}$$

can be used to describe a large number of interesting problems. Some specific examples are discussed in more detail in subsequent sections.

The above function θ is locally Lipschitzian and directionally differentiable—in the terminology of Robinson [30], θ is a *B(ouligand) differentiable* function. By the chain-rule for directional derivatives of a locally Lipschitzian function (see the cited reference), the directional derivative of θ at a vector $x \in R^n$ in the direction $d \in R^n$ is

$$\theta'(x, d) = h'(f(x), \nabla f(x)d).$$

Define the iteration function $\psi(x, d)$ by

$$\psi(x, d) = h(f(x) + \nabla f(x)d) - h(f(x)). \tag{21}$$

With this choice of $\psi(x, d)$, the objective function in the direction-finding problem (7) belongs to the class of "casting functions" used by Burke [2] for solving the problem (20). Note that the function $\psi(x, d)$ in (21) is, in general, neither positively homogeneous nor (globally) Lipschitz continuous in d. But $\psi(x, d)$ must be convex in d for each fixed x. Thus, the direction-finding problem (7) is a strictly convex program in d; in particular, (7) has a unique globally optimal solution. If, in addition, the function h is piecewise linear, then so is the iteration function $\psi(x, \cdot)$ for each fixed x; in this case, each subproblem (7) becomes a strictly convex piecewise quadratic program.

We now verify that the assumptions (A1)-(A4) are all satisfied by the choice of the iteration function given by (21).

Obviously, $\psi(x, d)$ is continuous in both arguments x and d, and $\psi(x, 0) = 0$ for all x. Thus, (A4) holds. Moreover, $\psi(x, d)$ majorizes the directional derivative $\theta'(x, d)$ by the convexity of the function h. Thus, (A1) holds. It remains to verify (A2) and (A3).

Fix the vector $\bar{x}$. Define the function $\psi_{\bar{x}}: R^n \to R$ by $\psi_{\bar{x}}(d) = \psi(\bar{x}, d)$. Then, condition (A2) will hold if for all $d \in R^n$

$$\psi'_{\bar{x}}(0, d) \leqq \theta'(\bar{x}, d). \tag{22}$$

By taking the directional derivative of the function $\psi_{\bar{x}}(d)$ at the zero vector and along a given direction d, we can see that the last inequality must hold as an equality. Thus, (A2) follows.

Finally, to establish (A3), let $\{z^k\}$ be a sequence converging to $\bar{x}$, $\{v^k\}$ be a sequence converging to $\bar{v}$, and $\{\lambda_k\}$ be a sequence of positive scalars converging to zero. Then,

$$\theta(z^k + \lambda_k v^k) - \theta(z^k) = h(f(z^k + \lambda_k v^k)) - h(f(z^k)) = h(y^k + \lambda_k u^k) - h(y^k)$$

where

$$y^k = f(z^k) \quad \text{and} \quad u^k = \nabla f(z^k)v^k + \frac{o(\lambda_k)}{\lambda_k}$$

and

$$\lim_{k\to\infty} \frac{o(\lambda_k)}{\lambda_k} = 0.$$

Clearly, the sequences $\{y^k\}$ and $\{u^k\}$ converge to $f(\bar{x})$ and $\nabla f(\bar{x})\bar{v}$, respectively. Since a convex function is subdifferentiably regular, it follows that

$$\limsup_{k\to\infty} \frac{\theta(z^k+\lambda_k v^k)-\theta(z^k)}{\lambda_k} \leqq h'(f(\bar{x}), \nabla f(\bar{x})\bar{v}).$$

Since the function $\psi(x, d)$ is continuous in both arguments, we have

$$\begin{aligned}\lim_{k\to\infty} \psi(z^k, v^k) &= \psi(\bar{x}, \bar{v}) \\ &= h(f(\bar{x})+\nabla f(\bar{x})\bar{v})-h(f(\bar{x})) \\ &\geqq h'(f(\bar{x}), \nabla f(\bar{x})\bar{v})\end{aligned}$$

where the last inequality follows because h is a convex function. Consequently, the required inequality (14) holds.

Summarizing the above derivation and invoking Theorem 1, we have established the following convergence result of the model algorithm specialized to minimize a composite convex function of the form (20).

COROLLARY 2. *Let $h: R^m \to R$ be a convex function bounded below on R^m, and let $f: R^n \to R^m$ be a continuously differentiable function. Let the iteration function $\psi(x, d)$ be defined by* (21) *and $\{B_k\}$ be a sequence of matrices satisfying assumption* (B). *Suppose that $\{x^k\}$ is a sequence of iterates produced by the model algorithm. Then, every accumulation point of $\{x^k\}$ is a stationary point of the composite function $\theta(x) = h(f(x))$.*

Remark. The bounded-below property of $\theta(x)$ follows from the same condition of $h(y)$.

3.3. The discrete minimax problem. An important special case of the composite convex minimization problem is the *discrete minimax problem* which has been studied extensively in the literature of nonsmooth optimization:

$$\text{(23)} \qquad \text{minimize} \quad \theta(x) := \max\,(f_i(x): i \in I), \qquad x \in R^n$$

where $I = \{1, \cdots, m\}$ and each $f_i: R^n \to R$ is continuously differentiable but not necessarily convex. In the notation of the composite convex minimization problem, we have

$$h(y) = \max\,(y_1, \cdots, y_m) \quad \text{and} \quad f(x) = (f_1(x), \cdots, f_m(x)).$$

The iteration function that results from the specialization of (21) to the discrete minimax problem (23) is given by

$$\psi_1(x, d) = \max_{i\in I}\,(f_i(x)+\nabla f_i(x)^T d)-\theta(x).$$

A noteworthy point about this function $\psi_1(x, d)$ is that the range of functions involved in the "max" operator is indexed by the whole set I. In the sequel, we define an alternative iteration function which takes into account only those component functions that do not deviate too significantly from the maximum. Specifically, let $\delta > 0$ be a given "tolerance" and define the index set $I_\delta(x)$ for each vector $x \in R^n$:

$$I_\delta(x) := \{i \in I: \theta(x) - f_i(x) < \delta\}.$$

Clearly, we have $I(x) \subseteq I_\delta(x)$, where

$$I(x) := \{i \in I: \theta(x) = f_i(x)\}$$

is the maximizing index set of $\theta(x)$. We define the iteration function:

$$\psi(x, d) = \max_{i \in I_\delta(x)} (f_i(x) + \nabla f_i(x)^T d) - \theta(x). \tag{24}$$

The use of an index set such as $I_\delta(x)$ is rather well known for the minimax problem (see [27]). Note that the resulting function $\psi(x, d)$ is piecewise linear and convex in d for each fixed x. In this case, the direction-finding subproblem (7) can be formulated as a convex quadratic program (see e.g., [14], [27]). Also note that, theoretically, any $\delta > 0$ in (24) will ensure that the method works, but there may be numerical reasons for not choosing δ to be too close to zero.

We verify that the assumptions (A1)-(A4) are satisfied. Clearly, for each fixed vector x, the function $\psi(x, d)$ is continuous in d, and $\psi(x, 0) = 0$. Since

$$\theta'(x, d) = \max_{i \in I(x)} (\nabla f_i(x)^T d),$$

and $I(x) \subseteq I_\delta(x)$, the function ψ majorizes the directional derivative θ'; thus (A1) holds. Unlike $\psi_1(x, d)$, the function $\psi(x, d)$ is generally not continuous in the variable x: an example will be provided below to illustrate this discontinuity. Thus, (A4) is a nontrivial condition.

Fix the vector $\bar{x}$. To establish (A2), we define the function $\psi_{\bar{x}}: R^n \to R$ by $\psi_{\bar{x}}(d) = \psi(\bar{x}, d)$. As in the case of the composite convex minimization problem, it suffices to verify the inequality (22). Again, by taking the directional derivative of the function $\psi_{\bar{x}}$ at the zero vector along a given direction d, it is not difficult to see that this required inequality must hold as an equality. Thus, (A2) is proved.

It remains to verify (A3) and (A4). The proof of (A3) is not difficult. Let $\{z^k\}$, $\{v^k\}$, and $\{\lambda_k\}$ be sequences as given in (A3). Then, since $\{z^k\}$ converges to $\bar{x}$, we must have for all k large enough, $I(\bar{x}) \subseteq I_\delta(z^k)$. Thus, the left-hand limit in (14) is at least as large as

$$\lim_{k \to \infty} \max_{i \in I(\bar{x})} [f_i(z^k) + \nabla f_i(z^k)^T v^k - \theta(z^k)] = \max_{i \in I(\bar{x})} [f_i(\bar{x}) + \nabla f_i(\bar{x})^T \bar{v} - \theta(\bar{x})] = \theta'(\bar{x}, \bar{v}),$$

where $\bar{v}$ is the limit of the sequence $\{v^k\}$. By the argument used in the previous case of a composite convex function, we see that the right-hand limsup in (14) is no greater than $\theta'(\bar{x}, \bar{v})$, thus, the desired condition (A3) is satisfied.

Finally, we verify condition (A4). For this purpose, let d be an arbitrary vector and $\{z^k\}$ be a sequence of vectors converging to $\bar{x}$ such that the limit

$$\lim_{k \to \infty} \psi(z^k, d)$$

exists. Since the index set I is finite, there must exist a subsequence $\{z^k: k \in K\}$ such that the index set $I_\delta(z^k)$ is the same for all $k \in K$. Call this common index set J. Then, we have

$$J \subseteq I_\delta(\bar{x}) \cup \tilde{I}_\delta(\bar{x})$$

where

$$\tilde{I}_\delta(\bar{x}) = \{i \in I: \theta(\bar{x}) - f_i(\bar{x}) = \delta\}.$$

Consequently, we have

$$\lim_{k\to\infty} \psi(z^k, d) = \lim_{k\to\infty,\, k\in K} [\max_{i\in J} (f_i(z^k) + \nabla f_i(z^k)^T d) - \theta(z^k)]$$

$$\leqq \max_{i\in I_\delta(\bar{x})\cup \bar{I}_\delta(\bar{x})} (f_i(\bar{x}) + \nabla f_i(\bar{x})^T d) - \theta(\bar{x})$$

$$= \max\left(\max_{i\in I_\delta(\bar{x})} (f_i(\bar{x}) + \nabla f_i(\bar{x})^T d - \theta(\bar{x})),\ \max_{i\in \bar{I}_\delta(\bar{x})} (\nabla f_i(\bar{x})^T d - \delta) \right).$$

Now, pick $\varepsilon > 0$ such that for all d with $\|d\| \leqq \varepsilon$,

$$\max_{i\in \bar{I}_\delta(\bar{x})} (\nabla f_i(\bar{x})^T d - \delta) \leqq \max_{i\in I_\delta(\bar{x})} (f_i(\bar{x}) + \nabla f_i(\bar{x})^T d - \theta(\bar{x})).$$

This is possible because, for all $i \in I_\delta(\bar{x})$, we have $f_i(\bar{x}) - \theta(\bar{x}) > -\delta$. Consequently, it follows that for such vectors d,

$$\lim_{k\to\infty} \psi(z^k, d) \leqq \psi(\bar{x}, d),$$

which is what assumption (A4) requires.

Summarizing the above derivation, we obtain the following result which establishes the convergence of our model descent algorithm specialized to the minimax problem (23).

COROLLARY 3. *Let $f_i: R^n \to R$ be continuously differentiable functions for $i \in I = \{1, \cdots, m\}$. Suppose that at least one function $f_i(x)$ is bounded below on R^n. Let the iteration function $\psi(x, d)$ be defined by* (24) *and $\{B_k\}$ be a sequence of matrices satisfying assumption* (B). *Suppose that $\{x^k\}$ is a sequence of iterates produced by the model algorithm. Then, every accumulation point of $\{x^k\}$ is a stationary point of the pointwise maximum function $\theta(x) = \max(f_i(x): i \in I)$.*

Remark. The bounded-below property of $\theta(x)$ follows from that of one of the f_i's.

The example below shows that the ε in assumption (A4) is essential in order for this condition to hold with the iteration function $\psi(x, d)$ given by (24).

Example. Consider three univariate functions

$$f_1(x) = x, \qquad f_2(x) = -x, \qquad f_3(x) = (x - \tfrac{1}{2})^2 + \tfrac{1}{2}.$$

Then,

$$\theta(x) = \max(|x|, (x - \tfrac{1}{2})^2 + \tfrac{1}{2}).$$

The unique global minimum point of θ is $\bar{x} = \frac{1}{2}$. Choose the scalar $\delta = 1$ and let the iteration function $\psi(x, d)$ be defined by (24). We claim that condition (A4) fails at the point $\bar{x}$ if there is no restriction on the size of d. Indeed, the limit condition in (A4) fails at $d = -2$ when $\{z^k\}$ is any sequence in the interval $[0, \frac{1}{2})$ converging to $\bar{x}$. We can easily verify that

$$\lim_{k\to\infty} \psi(z^k, -2) = 1 > 0 = \psi(\tfrac{1}{2}, -2).$$

3.4. The nonlinear complementarity problem. In the recent paper [15], we have presented a generalized Gauss-Newton method for solving a system of locally Lipschitzian equations. In what follows, we describe how the convergence theory developed for that method fits into the present framework of optimizing a locally Lipschitzian function. Rather than speaking in a general context, we focus our discussion on the *nonlinear complementarity problem* formulated as a system of B-differentiable equations. The two papers [25], [26] are also relevant to the treatment that follows.

The nonlinear complementarity problem is to find a vector $x \in R^n$ satisfying the conditions

$$x \geqq 0, \qquad f(x) \geqq 0, \qquad x^T f(x) = 0 \tag{25}$$

where $f: R^n \to R^n$ is a given continuously differentiable function. Define the function $H: R^n \to R^n$ by

$$H(x) = \min(x, f(x))$$

where the "min" operator denotes the componentwise minimum of two vectors. Consider the minimization problem (1) where

$$\theta(x) = \tfrac{1}{2} H(x)^T H(x)$$

is the *norm function* of H. The function H is B-differentiable, and so is θ. Our goal is to find a global minimizer $\bar{x}$ of the function θ; if $\theta(\bar{x}) = 0$, then $\bar{x}$ solves the problem (25).

In order to apply the model algorithm to minimize the norm function θ, we define the iteration function by

$$\psi(x, d) = H(x)^T G(x, d) \tag{26}$$

where the function $G: R^n \times R^n \to R^n$ is defined as (see [15]):

$$G_i(x, d) = \begin{cases} d_i & \text{if } x_i < f_i(x), \quad f_i(x) \geqq 0, \\ \nabla f_i(x)^T d & \text{if } x_i > f_i(x), \quad x_i \geqq 0, \\ \min(d_i, \nabla f_i(x)^T d) & \text{otherwise,} \end{cases}$$

for each $i = 1, \cdots, n$. Unlike the other iteration functions discussed so far, the function $\psi(x, d)$ given by (26) is positively homogeneous in d. Note that $\psi(x, d)$ is piecewise linear in d for each x. Moreover, if the vector x is such that there is no component i with $x_i = f_i(x) > 0$, then the function $\psi(x, d)$ is convex in d; in this case, the corresponding subproblem (7) can be easily reformulated as a convex quadratic program. In general, the function $\psi(x, d)$ belongs to the class of *quasi-differentiable functions* expressible as the difference of two piecewise linear convex functions (see [5] and the papers therein for more discussion of this class of nonsmooth functions).

As demonstrated in the reference, the function $\psi(x, d)$ defined above satisfies the assumptions (A1) and (A3). However, the remaining two assumptions (A2) and (A4) are not likely to hold without further conditions. As a substitute for these latter two conditions, we invoke the concept of regularity as defined in [25] and [26]. Specifically, we say that the vector $\bar{x}$ is *regular* if

(a) The matrix $\nabla_{\alpha_+} f_{\alpha_+}(\bar{x})$ is nonsingular;

(b) The Schur complement

$$\nabla_{\bar{\beta}} f_{\bar{\beta}}(\bar{x}) - \nabla_{\alpha_+} f_{\bar{\beta}}(\bar{x}) (\nabla_{\alpha_+} f_{\alpha_+}(\bar{x}))^{-1} \nabla_{\bar{\beta}} f_{\alpha_+}(\bar{x})$$

is a P-matrix.

Here, the index sets α_+ and $\bar{\beta}$ are given by

$$\alpha_+ = \{i: \bar{x}_i > f_i(\bar{x}), \bar{x}_i > 0\},$$

$$\bar{\beta} = \{i: \bar{x}_i = f_i(\bar{x})\} \cup \{i: \max(\bar{x}_i, f_i(\bar{x})) \leqq 0\},$$

and $\nabla_{\mathcal{I}} f_{\mathcal{J}}(\bar{x})$ denotes the partial Jacobian matrix $(\partial f_j(\bar{x})/\partial x_i)$ for $(i, j) \in \mathcal{I} \times \mathcal{J}$.

It has been proved in [26, Prop. 3] (see also [25, Prop. 3]) that if the vector $\bar{x}$ is regular, then there exists a neighborhood N of $\bar{x}$ so that all vectors $x \in N$ are also

regular. Moreover, if x is a regular vector, then the mapping $G_x: R^n \to R^n$ defined by $G_x(v) = G(x, v)$ is a homeomorphism of R^n onto itself. Finally, if $\bar{x}$ is regular, then for all vectors $x \in N$, the inverse mapping of G_x is Lipschitzian with the Lipschitz modulus independent of x in N. It is important to point out that the regularity of $\bar{x}$ implies these useful properties of vectors that are close to $\bar{x}$; vectors that are not within the given neighborhood N of $\bar{x}$ do not necessarily possess the stated properties.

We are now ready to state the following consequence of Theorem 1 when the model algorithm is specialized to solve the nonlinear complementarity problem (25).

COROLLARY 4. *Let $f: R^n \to R^n$ be a continuously differentiable function. Let the iteration function $\psi(x, d)$ be as defined by* (26) *and $\{B_k\}$ be a sequence of matrices satisfying assumption* (B). *Suppose that $\bar{x}$ is an accumulation point of a sequence $\{x^k\}$ produced by the model algorithm. If $\bar{x}$ is a regular vector, then $\bar{x}$ solves the nonlinear complementarity problem* (25).

Proof. Let $\{x^k: k \in K\}$ be a subsequence converging to $\bar{x}$. Let $\{d^k: k \in K\}$ be a corresponding sequence of directions generated by the algorithm. By carefully examining the proof of Theorem 1, it suffices to consider the case where the sequence $\{d^k: k \in K\}$ converges to zero: the absence of the assumptions (A2) and (A4) does not affect the rest of the argument in that proof.

Since $\bar{x}$ is regular, the iterate x^k is regular for all $k \in K$ sufficiently large. Thus, for each of these large enough $k \in K$, there exists a unique vector v^k satisfying

$$G(x^k, v^k) + H(x^k) = 0.$$

Then, for all scalar $\lambda > 0$, we have

$$\psi(x^k, \lambda v^k) = -\lambda H(x^k)^T H(x^k).$$

Moreover, since the sequence $\{H(x^k): k \in K\}$ converges, the sequence $\{v^k: k \in K\}$ is bounded (by the Lipschitzian property of the inverse of the mapping $G_{x^k}(\cdot)$ as noted above). Without loss of generality, we may assume that the latter sequence converges.

Since d^k is a globally optimal solution of the subproblem (7), we have for all k sufficiently large and all $\lambda > 0$,

$$\begin{aligned}\psi(x^k, d^k) + \frac{1}{2}(d^k)^T B_k d^k &\leqq \psi(x^k, \lambda v^k) + \frac{\lambda^2}{2}(v^k)^T B_k v^k \\ &= -\lambda H(x^k)^T H(x^k) + \frac{\lambda^2}{2}(v^k)^T B_k v^k.\end{aligned}$$

Passing to the limit as $\{k \to \infty, k \in K\}$, the first sum tends to zero as in the proof of Theorem 1, and the third sum converges to

$$-\lambda H(\bar{x})^T H(\bar{x}) + \frac{\lambda^2}{2}\bar{v}\bar{B}\bar{v}.$$

Now, dividing by λ and passing to the limit as $\lambda \downarrow 0$, we immediately obtain $H(\bar{x}) = 0$, which implies that $\bar{x}$ is a desired solution of the problem (25). □

We should point out that although the two conditions (A2) and (A4) are not explicitly used in the above proof, they are implied by the regularity assumption at $\bar{x}$. The reason for this is that under the regularity assumption, we know that $\bar{x}$ is a solution of the nonlinear complementarity problem (25). Using this latter fact, we can easily show that (A2) and (A4) must hold at $\bar{x}$. The details are omitted.

3.5. The generalized feasibility problem. Another important instance of a composite convex minimization problem is the *generalized feasibility problem.* The latter problem is to find a vector $x \in R^n$ such that

$$g(x) \in K \tag{27}$$

where K is a closed convex set in R^m and $g: R^n \to R^m$ is continuously differentiable. The recent paper [3] discusses a special case of this problem with K being a cone.

In order to cast the problem (27) in the form of a composite convex minimization problem, define the *distance function* $\rho: R^m \to R$ by

$$\rho(y) := \operatorname{dist}(y|K) = \min\{\|z - y\|_2 : z \in K\}$$

where $\|\cdot\|_2$ is the Euclidean norm on R^m. Notice that ρ is well defined because any vector in R^m has a projection onto a closed convex set. Let $\theta: R^n \to R$ be the composite function

$$\theta(x) = \rho \circ g(x).$$

Clearly, θ is a nonnegative function. Moreover, the problem of finding a vector x such that $g(x) \in K$ can be solved by computing a global minimum point $\bar{x}$ of $\theta(x)$ (if it exists) and by checking if $\theta(\bar{x}) = 0$. In turn, the problem of minimizing the function θ is a composite convex minimization problem because ρ is a convex function.

Now the theory of composite convex minimization described in § 3.2 can be applied. In particular, by defining the iteration function ψ to be

$$\psi(x, d) = \rho(g(x) + \nabla g(x)d) - \rho(g(x)),$$

we obtain a descent algorithm for computing a stationary point of the composite function $\theta(x) = \rho(g(x))$.

In [3], Burke and Han used a slightly different subproblem to find a search direction d^k. Instead of the positive-definite matrix B_k, they employed what amounts to the selection of a minimum-norm solution of their subproblem. Because of the positive definiteness of the matrix B_k and the convexity of the function $\psi(x, d)$ in the variable d, our subproblem (7) has a unique optimal solution. The linesearch routine used in the cited reference is very similar in spirit to the one used here.

4. Mathematical programs with equilibrium constraints. We now come to the final application of our general theory, which concerns a mathematical program with equilibrium constraints. This problem appears in various forms and in many applications. Our terminology follows the one used in the paper [16].

In its most general form, a mathematical program with equilibrium constraints can be defined as follows. Let $Y: R^n \to R^m$ be a given multifunction; i.e., for each $x \in R^n$, $Y(x)$ is a nonempty subset of R^m. Let $\mathcal{G}_Y$ denote the graph of Y, i.e.,

$$\mathcal{G}_Y = \{(x, y) \in R^n \times R^m : y \in Y(x)\}.$$

Let $f: R^n \times R^m \to R$ be a given continuously differentiable function. The *mathematical program with equilibrium constraints* is:

$$\begin{aligned} &\text{minimize} && f(x, y) \\ &\text{subject to} && (x, y) \in \mathcal{G}_Y, \qquad x \in X \end{aligned} \tag{28}$$

where X is a closed convex set in R^n. The reason why the term "equilibrium constraints" is attached to this problem is that in many of its applications, $Y(x)$ represents the solution set of a (parametric) *equilibrium programming problem* which is used to model

certain equilibrium (or optimality) conditions. For example, in the traffic network design problem [21], [11], $Y(x)$ denotes the set of *user equilibrium flows*, x represents the design variable of arc capacities, and the objective function $f(x, y)$ represents the system design cost; in the Stackelberg leader-follower game problem [33], $Y(x)$ is the *response function* of the followers who act as the players in a classical Cournot-Nash equilibrium game, x is the decision variable of the leader, and $f(x, y)$ is the overall objective function that the leader wishes to minimize. In both these applications, $Y(x)$ is the solution set of a variational inequality problem parametrized by the primary variable x. Another important special case of (28) has $Y(x)$ equal to the optimal solution set of a nonlinear program dependent on the parameter x; this latter problem is commonly known as the *bilevel program* [1].

4.1. The implicit programming problem. The general mathematical programming problem with equilibrium constraints is very difficult: the multivalued nature of the mapping Y is a major factor. In the sequel, we consider the special case of (28) in which $X = R^n$ and $Y(x)$ is a singleton for each x, say, $Y(x) := \{y(x)\}$. Specifically, we study the (unconstrained) *implicit programming problem*:

$$\text{minimize} \quad \theta(x) := f(x, y(x)) \tag{29}$$

where $y: R^n \to R^m$ is a given single-valued function and $f: R^n \times R^m \to R$ is continuously differentiable. The implicit nature of the problem (29) is due to the fact that the function $y(x)$ is known only implicitly. Our goal is to apply the model algorithm to compute a stationary point of the problem (29).

In order to somewhat simplify the notation and the discussion, we assume that for each $x \in R^n$, $y(x)$ is the unique solution of the parametric nonlinear complementarity problem:

$$y \geqq 0, \qquad F(x, y) \geqq 0, \qquad y^T F(x, y) = 0 \tag{30}$$

where $F: R^n \times R^m \to R^m$ is continuously differentiable. Our results can be easily extended to the case where $y(x)$ is the solution of a variational inequality problem or a nonlinear program parametrized by x.

Under a regularity assumption, the solution function $y(x)$ is actually B-differentiable, thus, so is the objective function $\theta(x)$. It would be useful for us to summarize the differentiability properties of $y(x)$. For this purpose, we introduce three basic index sets associated with a solution of the nonlinear complementarity problem (30):

$$\alpha(x) = \{i : y_i(x) > 0 = F_i(x, y(x))\},$$
$$\beta(x) = \{i : y_i(x) = 0 = F_i(x, y(x))\},$$
$$\gamma(x) = \{i : y_i(x) = 0 < F_i(x, y(x))\}.$$

The following result describes the local behavior of a solution $y(\bar{x})$ of the problem (30) at a given vector $\bar{x}$ under the assumption of regularity. The proof of this result can be found in [29] and [24].

PROPOSITION 3. *Let $F: R^n \times R^m \to R^m$ be continuously differentiable. Let $\bar{y}$ be a solution of the problem* (30) *corresponding to $\bar{x}$. Suppose that $\bar{y}$ is regular, i.e.,*

(a) *The matrix $\nabla_{y_\alpha} F_\alpha(\bar{x}, \bar{y})$ is nonsingular; and*

(b) *The Schur complement*

$$\nabla_{y_\beta} F_\beta(\bar{x}, \bar{y}) - \nabla_{y_\alpha} F_\beta(\bar{x}, \bar{y})(\nabla_{y_\alpha} F_\alpha(\bar{x}, \bar{y}))^{-1} \nabla_{y_\beta} F_\alpha(\bar{x}, \bar{y})$$

is a P*-matrix.* (*The dependence on the vector $\bar{x}$ is omitted from the index sets $\alpha(\bar{x})$ and $\beta(\bar{x})$ in the above two and the subsequent conditions.*)

Then, there exist neighborhoods U of $\bar{x}$ and V of $\bar{y}$ and a Lipschitzian function $y: U \to V$ such that $y(\bar{x}) = \bar{y}$ and

(i) *For each $x \in U$, $y(x)$ is the unique solution of the problem* (30) *in V;*

(ii) *The function $y(\cdot)$ is directionally differentiable at $\bar{x}$; the directional derivative $y'(\bar{x}, d)$ (for $d \in R^n$) is the unique solution $v \in R^m$ of the following mixed linear complementarity system:*

$$
\begin{aligned}
&\nabla_x F_\alpha(\bar{x}, y(\bar{x}))d + \nabla_{y_\alpha} F_\alpha(\bar{x}, y(\bar{x}))v_\alpha + \nabla_{y_\beta} F_\alpha(\bar{x}, y(\bar{x}))v_\beta = 0,\\
&\nabla_x F_\beta(\bar{x}, y(\bar{x}))d + \nabla_{y_\alpha} F_\beta(\bar{x}, y(\bar{x}))v_\alpha + \nabla_{y_\beta} F_\beta(\bar{x}, y(\bar{x}))v_\beta \geqq 0,\\
&v_\alpha \text{ unrestricted}, \quad v_\gamma = 0, \quad v_\beta \geqq 0,\\
&(v_\beta)^T[\nabla_x F_\beta(\bar{x}, y(\bar{x}))d + \nabla_{y_\alpha} F_\beta(\bar{x}, y(\bar{x}))v_\alpha + \nabla_{y_\beta} F_\beta(\bar{x}, y(\bar{x}))v_\beta] = 0;
\end{aligned} \tag{31}
$$

(iii) *The function $y(\cdot)$ is strongly* F*-differentiable at $\bar{x}$ if and only if either $\beta(\bar{x}) = \varnothing$ or*

$$
\nabla_x F_\beta(\bar{x}, y(\bar{x})) - \nabla_{y_\alpha} F_\beta(\bar{x}, y(\bar{x}))(\nabla_{y_\alpha} F_\alpha(\bar{x}, y(\bar{x})))^{-1}\nabla_x F_\alpha(\bar{x}, y(\bar{x})) = 0;
$$

(iv) *If $\nabla y(\bar{x})$ exists, then it is given by*

$$
\nabla y_\alpha(\bar{x}) = -(\nabla_{y_\alpha} F_\alpha(\bar{x}, y(\bar{x})))^{-1}\nabla_x F_\alpha(\bar{x}, y(\bar{x})), \qquad \nabla y_{\beta\cup\gamma}(\bar{x}) = 0.
$$

As a consequence of part (iii) in the above result, we note that if *strict complementarity* holds throughout the complementarity problem (30), that is, if $y(x) + F(x, y(x)) > 0$ for all x, then the solution function $y(x)$ is continuously differentiable on R^n, and the problem (29) becomes a smooth optimization problem. In essence, this strict complementarity property is what is assumed in many heuristic methods for solving the problem (30), see, e.g., [11]. By using the directional derivative $y'(x, d)$, we can therefore considerably relax this rather restrictive F-differentiability assumption of the solution function $y(x)$.

We should point out that as a function in the direction d, the directional derivative $y'(x, d)$ is piecewise linear (for fixed x). This follows from the observation that the system (31) is a parametric mixed linear complementarity problem with d as the parameter.

4.2. Direction finding step. With the directional derivative $y'(x, d)$ of the solution function $y(x)$ given in Proposition 3, we define the iteration function $\psi(x, d)$ to be the directional derivative of θ. By the chain rule for the directional derivative, we obtain

$$
\psi(x, d) = \theta'(x, d) = \nabla_x f(x, y(x))d + \nabla_y f(x, y(x))y'(x, d),
$$

which shows, among other things, that $\psi(x, d)$ remains a piecewise linear function in d.

With the iteration function $\psi(x, d)$ given as above, the direction-finding subproblem (7) can be stated in the following form: (by letting $y^k = y(x^k)$)

$$
\begin{aligned}
&\text{minimize} \quad \nabla_x f(x^k, y^k)d + \nabla_y f(x^k, y^k)v + \tfrac{1}{2}d^T B_k d\\
&\text{subject to} \quad v = y'(x^k, d).
\end{aligned} \tag{32}
$$

This is a problem of minimizing a convex quadratic function subject to linear contraints and possibly some linear complementarity conditions. If the index set $\beta(x^k)$ is empty (in which case, the solution function $y(x)$ is F-differentiable at x^k), (32) reduces to an unconstrained strictly convex quadratic program in the variable d only. If the cardinality of the index set $\beta(x^k)$ is small, ($\beta(x^k)$ consists of the *degenerate indices*), then the number of complementarity constraints present in (32) is correspondingly

small; in this case, the problem (32) could be solved effectively by either complete enumeration or a branch-and-bound scheme (see, e.g., [13]). In general, the direction-finding problem (32) is potentially much simpler to solve than the original problem (29) and seems to be a reasonable subproblem on which to base an iterative procedure.

4.3. Convergence and some computational results. The convergence of the resulting algorithm for solving the problem (29) follows directly from Corollary 1. The strong F-differentiability assumption of θ at the vector $\bar{x}$ (which is required in the corollary) is satisfied if the solution function $y(x)$ has a strong F-derivative at $\bar{x}$; according to Proposition 3, this in turn is true if strict complementarity holds at the solution $y(\bar{x})$. The noteworthy point here is that such a strong differentiability assumption is needed only at the limit vector $\bar{x}$.

We have obtained some preliminary computational results with the above iterative scheme for solving some network design problems discussed in [11] and some bilevel programs from [7]. The results suggest that the scheme is quite effective and requires very few iterations. During the solution process, we encounter some nondifferentiable points which are handled as described. More details of the implementation and the results are documented in the thesis [28].

5. Nonmonotone linesearch. In this and the next section, we discuss two modifications of the basic algorithm presented in § 2. The first involves a nonmonotone linesearch technique which extends the basic Armijo condition (11). The second modification deals with a constrained version of the basic problem (1).

Grippo, Lampariello, and Lucidi [12] have discussed a nonmonotone linesearch technique for Newton's method to solve smooth unconstrained optimization problems. Recently, Ferris, Grippo, and Lucidi [8] have applied this technique to solve the nonlinear complementarity problem formulated as a system of nonlinear equations. The nonmonotone linesearch idea needs only some slight modification to remain valid in the present context of nonsmooth problems. In what follows, we describe the modified algorithm and establish its convergence. Subsequently, the significance and usefulness of this technique will be briefly discussed.

5.1. The modified algorithm and its convergence. We consider the unconstrained problem (1). The algorithm essentially works in the same way as before except that the Armijo rule (11) is modified. More specifically, given the iterate x^k, the direction-finding step still relies on the availability of the iteration function ψ, and consists of solving the same subproblem (7). If the computed direction is d^k, the linesearch step is modified as follows.

Choose σ and $\rho \in (0, 1)$ and a nonnegative integer M. For each k, let $m(k)$ be chosen such that

$$m(0) = 0, \qquad 0 \leqq m(k) \leqq \min [m(k-1)+1, M] \quad \text{for } k \geqq 1.$$

Now, let

$$x^{k+1} = x^k + \tau_k d^k$$

where $\tau_k = \rho^{m_k}$ with m_k being the first nonnegative integer m such that

$$\theta(x^k + \rho^m d^k) - \max_{0 \leqq j \leqq m(k)} [\theta(x^{k-j})] \leqq -\frac{\sigma}{2} \rho^m (d^k)^T B_k d^k. \tag{33}$$

The modified Armijo condition (33) says that we should maintain sufficient decrease in the objective function, not necessarily with respect to the most recent value

$\theta(x^k)$, but with respect to the largest of the previous $m(k)+1$ values. A simple choice of $m(k)$ is to let $m(k)=k$, for $k \leqq M$ and to let $m(k)=M$ thereafter. Typical values for M are between 5 and 10.

We can now state the convergence result for the model algorithm with non-monotone linesearch. The proof will only be sketched. A similar result was proved recently in [8].

THEOREM 2. *Suppose that the level set* (13) *is bounded. Then, Theorem* 1 *remains valid with the above modified linesearch routine.*

Proof. For each k, let $l(k)$ be such that

$$k-m(k) \leqq l(k) \leqq k \quad \text{and} \quad \theta(x^{l(k)}) = \max_{0 \leqq j \leqq m(k)} [\theta(x^{k-j})].$$

Since $m(k)$ is bounded, it follows that $l(k) \to \infty$ as k does. We have (see [12])

$$\theta(x^{l(k+1)}) \leqq \theta(x^{l(k)}).$$

Since θ is bounded below, it follows that the sequence $\{\theta(x^{l(k)})\}$ converges. From the linesearch rule (33), we have, for $k > M$,

$$\theta(x^{l(k)}) - \theta(x^{l(l(k)-1)}) \leqq -\frac{\sigma}{2} \tau_{l(k)-1} (d^{l(k)-1})^T B_k d^{l(k)-1}.$$

Since the sequence $\theta(x^{l(k)})$ has a limit and assumption (B) is in force, we deduce

$$\lim_{k \to \infty} \tau_{l(k)-1} d^{l(k)-1} = 0.$$

As in [12], it can be shown by induction that if

$$l'(k) := l(k+M+2)$$

then, for any given $j \geqq 1$

$$\lim_{k \to \infty} \tau_{l'(k)-j} d^{l'(k)-j} = 0 \quad \text{and} \quad \lim_{k \to \infty} \theta(x^{l'(k)-j}) = \lim_{k \to \infty} \theta(x^{l(k)}). \tag{34}$$

For any k, the following identity holds:

$$x^{k+1} + \sum_{j=1}^{l'(k)-(k+1)} \alpha_{l'(k)-j} d^{l'(k)-j} = x^{l'(k)}.$$

By the fact that $l'(k)-k-1 = l(k+M+2)-k-1 \leqq M+1$ and by (34), we get

$$\lim_{k \to \infty} (x^{k+1} - x^{l'(k)}) = 0. \tag{35}$$

Since the level set (13) is bounded and contains the entire sequence of iterates $\{x^k\}$, and since θ is continuous, it follows that θ is uniformly continuous on that level set. Hence, the entire sequence $\{\theta(x^k)\}$ converges because the subsequence $\{\theta(x^{l(k)})\}$ does. This yields, by (33),

$$\lim_{k \to \infty} \tau_k d^k = 0.$$

From this point on, the proof of Theorem 1 applies. We need only to observe that the condition (18) continues to hold because $\theta(x^k) \leqq \theta(x^{l(k)})$. □

5.2. Computational significance. A potential computational advantage of the non-monotone linesearch technique is that it permits larger steps to be taken at intermediate stages of the minimization process. This is accomplished by allowing the objective

value $\theta(x^{k+1})$ to increase from the most recent one $\theta(x^k)$, but still ensuring a sufficient decrease from some previous value $\theta(x^{l(k)})$. The end effect of this technique is that it could reduce the number of function evaluations at each linesearch step (but in general, not necessarily the number of total linesearches) while maintaining the overall global convergence of the method.

The computational results reported in [12] suggest that in the context of Newton's method for solving smooth unconstrained problems, the numbers of linesearches and function evaluations are both considerably smaller with the nonmonotone linesearch technique included, than those with the usual Armijo rule. It is hoped that the same improvement can be achieved for nonsmooth problems. Computational testing is presently being carried out and the results will be reported in [28].

The reduction in the number of function evaluations is especially useful when the function itself is complicated and expensive to evaluate. This is the case, for example, in the implicit programming problem (29) where each evaluation of the function $y(x)$ typically requires the solution of a nonlinear program, a nonlinear complementarity, or a variational inequality problem.

6. Constrained problems. In this final section, we briefly discuss the extension of the model algorithm to deal with the following linearly constrained nonsmooth optimization problem:

$$\text{minimize} \quad \theta(x): x \in \Omega \tag{36}$$

where Ω is a polyhedron in R^n. As before, the function $\theta: R^n \to R$ is assumed to be locally Lipschitzian. We say that a feasible vector $\bar{x} \in \Omega$ is a *Dini stationary point* of the problem (36) if

$$\theta^D(\bar{x}, y - \bar{x}) \geqq 0 \quad \text{for all } y \in \Omega.$$

In essence, the algorithm described below is a *feasible-point descant* method for finding a Dini stationary point of (36). Since this is a feasible-point method, all iterates generated will be feasible to the problem (36). For computational reasons, we require the feasible region Ω to be polyhedral in order for the direction-finding problem (see below) to be practically implementable. As far as the convergence theory is concerned, it is enough for Ω to be a closed convex set in R^n.

The algorithm still consists of a direction-finding step followed by a linesearch step (incorporating the nonmonotone linesearch technique of the previous section is certainly possible but is not adopted in the sequel). We continue to depend on an appropriately chosen iteration function $\psi(x, d)$, and we shall impose a modified set of assumptions on $\psi(x, d)$ which takes into account the presence of the feasible region Ω. The following is a detailed description of the modified algorithm for computing a Dini stationary point of (36).

The Model Algorithm for constrained problem. Let $\rho, \sigma \in (0, 1)$ be given. Let $x^0 \in \Omega$ be arbitrary. Set $k = 0$. In general, given $x^k \in \Omega$, let y^k be an (arbitrary) globally optimal solution of the following problem:

$$\text{minimize} \quad \psi(x^k, y - x^k) + \tfrac{1}{2}(y - x^k)^T B_k (y - x^k): y \in \Omega. \tag{37}$$

Terminate if the optimum objective value of (37) is zero; in this case, x^k is a desired constrained Dini stationary point of θ (see Theorem 3 below). Otherwise, let $d^k = y^k - x^k$ be the search direction and let x^{k+1} be generated as in the model algorithm of § 2. Test x^{k+1} for convergence. Repeat the general step with $k+1$ replacing k if x^{k+1} fails the convergence test.

There are two new elements in the above algorithm; both are included as a result of the presence of the constraint set Ω. One is the requirement that the initial iterate x^0 be a feasible vector of the problem (36); the other is the subproblem (37) which is now constrained by the set Ω. Note that the next iterate x^{k+1} is ensured feasible to (36) because x^{k+1} lies on the line segment joining the two feasible vectors x^k and y^k.

The convergence of the above modified algorithm—including the finite termination situation—can be established by an argument similar to that in the unconstrained case. Three of the assumptions may be relaxed to hold only for the "feasible vectors." Specifically, the modification of (A1) is as follows.

(A1′) For each fixed vector $x \in \Omega$, the function $\psi(x, d)$ is continuous in the variable d, and $\psi(x, 0) = 0$. Moreover, for all $(x, y) \in \Omega \times \Omega$,

$$\psi(x, y-x) \geqq \theta^D(x, y-x).$$

The modification of assumptions (A2) and (A4) also concerns the limit point $\bar{x}$ whose stationarity is the issue at hand.

(A2′) For every $y \in \Omega$,

$$\liminf_{\lambda \to 0_+} \frac{\psi(\bar{x}, \lambda(y-\bar{x}))}{\lambda} \geqq \theta^D(\bar{x}, y-\bar{x}).$$

(A4′) There exists a scalar $\varepsilon > 0$ such that for every vector $y \in \Omega$ satisfying $\|y - \bar{x}\| \leqq \varepsilon$ and every sequence $\{z^k\}$ converging to $\bar{x}$,

$$\liminf_{k \to \infty} \psi(z^k, y-\bar{x}) \geqq \psi(\bar{x}, y-\bar{x}).$$

Under the above modified assumptions, we state the principal convergence result for the modified descent algorithm for computing a Dini stationary point of the problem (36).

THEOREM 3. *Let Ω be a polyhedron in R^n and $\theta: R^n \to R$ be a locally Lipschitzian function that is bounded below on Ω. Let $\psi: R^n \times R^n \to R$ and $\{B_k\}$ also be given. Suppose that assumptions* (A1′) *and* (B) *hold. Then, if $\bar{x}$ is an accumulation point of a sequence $\{x^k\}$ produced by the above modified algorithm, and if the assumptions* (A2′), (A3), *and* (A4′) *holds at $\bar{x}$, then $\bar{x}$ is a Dini stationary point of the constrained problem* (36).

Proof. Since the proof is very similar to the unconstrained case, we only establish the stated conclusion in the finite termination situation. Specifically, we demonstrate that if the modified direction-finding subproblem (37) has a zero optimum objective value, then the current iterate ^{k} is a Dini stationary point of (36), provided that x^k satisfies the assumption (A2′), i.e., if for all $y \in \Omega$,

$$\liminf_{\lambda \to 0_+} \frac{\psi(x^k, \lambda(y-x^k))}{\lambda} \geqq \theta^D(x^k, y-x^k) \tag{38}$$

(cf. Proposition 2). So, suppose that (37) has a zero objective value. Let $y \in \Omega$ and $\lambda \in (0, 1)$ be arbitrary. Then the vector $x^k + \lambda(y-x^k) \in \Omega$. Hence, we have

$$0 \geqq \psi(x^k, \lambda(y-x^k)) + \frac{\lambda^2}{2}(y-x^k)^T B_k (y-x^k).$$

Dividing by λ, passing to the limit $\lambda \downarrow 0$, and using the assumed inequality (38), we

conclude that for all $y \in \Omega$,

$$0 \leqq \theta^D(x^k, y - x^k)$$

as desired. □

REFERENCES

[1] J. BARD, *An algorithm for solving the general bi-level programming problem*, Math. Oper. Res., 8 (1983), pp. 260-272.

[2] J. V. BURKE, *Descent methods for composite nondifferentiable optimization problems*, Math. Programming, 33 (1985), pp. 260-279.

[3] J. V. BURKE AND S. P. HAN, *A Gauss-Newton approach to solving generalized inequalities*, Math. Oper. Res., 11 (1986), pp. 632-643.

[4] F. H. CLARKE, *Optimization and Nonsmooth Analysis*, John Wiley, New York, 1983.

[5] V. F. DEM'YANOV AND L. C. W. DIXON, EDS., *Quasidifferentiable calculus*, Math. Programming Stud., 29 (1986), pp. 20-43.

[6] V. F. DEM'YANOV AND L. V. VASIL'EV, *Nondifferentiable Optimization*, Optimization Software, Inc., New York, 1985.

[7] A. H. DESILVA, *Sensitivity formulas for nonlinear factorable programming and their applications to the solution of an implicitly defined optimization model of United States crude oil production*, Ph.D. thesis, Department of Operations Research, The George Washington University, Washington, DC, 1978.

[8] M. C. FERRIS, L. GRIPPO, AND S. LUCIDI, *Globally convergent methods for nonlinear complementarity problems*, paper presented at the Joint Operations Research Society of America/Institute of Management Sciences National Meeting, Las Vegas, NV, May 7-9, 1990.

[9] R. FLETCHER, *A model algorithm for composite nondifferentiable optimization problems*, Math. Programming Stud., 17 (1982), pp. 67-76.

[10] ———, *Practical Methods of Optimization*, Second Edition, John Wiley, New York, 1987.

[11] T. L. FRIESZ, R. L. TOBIN, H. J. CHO, AND N. J. MEHTA, *Sensitivity analysis-based heuristic algorithms for mathematical programs with variational inequality constraints*, Math. Programming, B, to appear.

[12] L. GRIPPO, F. LAMPARIELLO, AND S. LUCIDI, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707-716.

[13] W. P. HALLMAN, *Complementarity in mathematical programming*, Ph.D. thesis, Department of Industrial Engineering, University of Wisconsin, Madison, WI, 1979.

[14] S. P. HAN, *Variable metric methods for minimizing a class of non-differentiable functions*, Math. Programming, 20 (1981), pp. 1-13.

[15] S. P. HAN, J. S. PANG, AND N. RANGARAJ, *Globally convergent newton methods for nonsmooth equations*, Math. Oper. Res., to appear.

[16] P. T. HARKER AND J. S. PANG, *Existence of optimal solutions to mathematical programs with equilibrium constraints*, Oper. Res. Lett., 7 (1988), pp. 61-64.

[17] K. C. KIWIEL, *Methods of descent for nondifferentiable optimization*, Lecture Notes in Mathematics 1133, Springer-Verlag, Berlin, 1985.

[18] C. LEMARECHAL, *Nonsmooth optimization and descent methods*, Res. Report RR-78-4, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1978.

[19] ———, *Bundle methods in nonsmooth optimization*, in Nonsmooth Optimization, C. Lemarechal and R. Mifflin, eds., Pergamon Press, Oxford, 1978.

[20] C. LEMARECHAL AND R. MIFFLIN, EDS., *Nonsmooth Optimization*, Pergamon Press, Oxford, 1978.

[21] P. MARCOTTE, *Network design problem with congestion effects: A case of bilevel programming*, Math. Programming, 34 (1986), pp. 142-162.

[22] R. MIFFLIN, *An algorithm for constrained optimization with semismooth functions*, Math. Oper. Res., 2 (1977), pp. 191-207.

[23] ———, *A modification and an extension of Lemarechal's algorithm for nonsmooth minimization*, Math. Programming Stud., 17 (1982), pp. 77-90.

[24] J. S. PANG, *Two characterization theorems in complementarity theory*, Oper. Res. Lett., 7 (1988), pp. 27-31.

[25] ———, *Newton's method for B-differentiable equations*, Math. Oper. Res., 15 (1990), pp. 311-341.

[26] ———, *A B-differentiable equation based, globally and locally quadratically convergent algorithm for nonlinear programs, complementarity and variational inequality problems*, Math. Programming, to appear.

[27] B. N. PSHENICHNY AND Y. M. DANILIN, *Numerical Methods in Extremal Problems*, MIR, Moscow, 1978. (English translation.)

[28] N. RANGARAJ, *Nonsmooth optimization: Algorithms and applications*, Ph.D. thesis, Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, MD, 1990.

[29] S. M. ROBINSON, *Strongly regular generalized equations*, Math. Oper. Res., 5 (1980), pp. 43–62.

[30] ———, *Local structure of feasible sets in nonlinear programming, Part* III: *Stability and sensitivity*, Math. Programming Stud., 30 (1987), pp. 45–66.

[31] R. T. ROCKAFELLAR, *Computational schemes for large-scale problems in extended linear-quadratic programming*, Math. Programming, B, to appear.

[32] R. T. ROCKAFELLAR AND J. SUN, *A simplex-active-set algorithm for piecewise quadratic programming*, Tech. Report 86-10, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1986.

[33] H. D. SHERALI, A. L. SOYSTER, AND F. H. MURPHY, *Stackelberg-Nash-Cournot equilibria: Characterization and computation*, Oper. Res., 31 (1983), pp. 253–276.

[34] J. SUN, *Basic theories and selected applications of monotropic piecewise quadratic programming*, Tech. Report 86-09, Department of Industrial Engineering and Mangement Sciences, Northwestern University, Evanston, IL, 1986.

[35] N. Z. SHOR, *Minimization Methods for Nondifferentiable Functions*, Springer-Verlag, Berlin, 1985. (English translation.)

[36] Y. YUAN, *Conditions for convergence of trust region algorithms for nonsmooth optimization*, Math. Programming, 31 (1985), pp. 220–228.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 83-92, February 1991

A POLYNOMIAL-TIME PREDICTOR-CORRECTOR ALGORITHM FOR A CLASS OF LINEAR COMPLEMENTARITY PROBLEMS*

JIU DING†‡ AND TIEN-YIEN LI†

Abstract. A polynomial-time algorithm for a class of linear complementarity problems with positive semidefinite matrices is presented. The method is based on a one-step Euler's prediction and one-step Newton's correction procedure to follow the homotopy path defined as the set $\{(x, y) \in \mathbb{R}^n_+ : x_i y_i = \mu, i = 1, \cdots, n, \mu > 0, y = Mx + q\}$, and solves the problem in $O(n^{3.5}L)$ arithmetic operations. Moreover, after one iteration the value $x^T y$ decreases with the ratio at least $(1 - (2/5\sqrt{n}))$.

Key words. linear complementarity problem, polynomial-time algorithm, predictor-corrector method

AMS(MOS) subject classifications. 90, 49

1. Introduction. Let $M \in \mathbb{R}^{n \times n}$ be an $n \times n$ real matrix and $q \in \mathbb{R}^n$ be an n-dimensional real vector. Here $\mathbb{R}^n$ has the usual Euclidean spectral norm and $\mathbb{R}^{n \times n}$ has the corresponding operator norm. Consider the linear complementarity problem: finding $(x, y) \in \mathbb{R}^{2n}$ satisfying

$$\text{(LCP)} \qquad y = Mx + q, \qquad (x, y) \geqq 0, \qquad x^T y = 0.$$

A traditional approach to this problem is the pivoting method developed by Lemke and others [15]. Like the simplex method for linear programming, this method searches the solution point along the boundary of some polyhedral. In the worst case, an exponential number of pivoting operations is required. With the appearance of Karmarkar's pioneering work [8], the path-following interior point methods have attracted tremendous attentions in the field of mathematical programming. See [6], [7], [11], [13], [14], [16], [18], [19], and the references therein. For the (LCP), Megiddo [12] set up a theoretical framework for the path-following approaches. Kojima, Mizuno, and Yoshise [9] first gave a practical polynomial-time interior point algorithm for a class of linear complementarity problems. In their algorithms, a one-step Newton's iteration is used at each cycle with a new updated parameter value to follow the homotopy path guaranteed by the classic implicit function theorem. In their $O(n^{3.5}L)$ algorithm, it is proved that the decreasing rate at each iteration is at least $(1 - (1/8\sqrt{n}))$ with $O(n^3)$ arithmetic operations.

Currently, most of the path-following interior point methods in convex programming have the same feature: for an appropriate mapping $F : \mathbb{R}^n \times \mathbb{R}_+ \to \mathbb{R}^n$ and an initial point x^0, recursively calculate

$$x^{i+1} = x^i - F_x(x^i, t^{i+1})^{-1} F(x^i, t^{i+1})$$

for some sequence $\{t^i\}$ monotonically approaching zero [17]. Thus the path is followed in a zigzag way to achieve a polynomial-time complexity. In terms of general predictor-corrector homotopy continuation path-following algorithms [1], this feature amounts to a zero-order prediction and one-step Newton correction. In this paper, we follow

* Received by the editors May 7, 1990; accepted for publication (in revised form) August 21, 1990. This research was supported in part by National Science Foundation grant DMS-8902663.

† Department of Mathematics, Michigan State University, East Lansing, Michigan 48824.

‡ Present address, Department of Mathematics, University of Southern Mississippi, Hattiesburg, Mississippi 39406-5045.

the path by using Euler's method as first-order "predictor" followed by one-step Newton's method as "corrector." Although we need to solve two systems of linear equations with $2n^3$ arithmetic operations at each cycle, the rate of decrease of $x^T y$, which characterizes the extent of complementarity of x and y obtained after each cycle, is at least $(1-(2/5\sqrt{n}))$, 1.6 times as large as that of Kojima, Mizuno, and Yoshise's $O(n^{3.5}L)$ algorithm for the coefficient before $1/\sqrt{n}$ after two cycles.

The paper is organized as follows. The algorithm is described in § 2. The convergence theorem as well as the bounds of complexity of our algorithm are presented in § 3. Some further comments are given in § 4.

2. A predictor-corrector algorithm. Throughout this paper, we assume that the matrix $M \in \mathbb{R}^{n\times n}$ in (LCP) is positive semidefinite (not necessarily symmetric) with no zero rows and $n \geqq 3$. Denote by $\mathbb{R}^n_+$ and $\mathbb{R}^n_{++}$ the nonnegative orthant $\{x \in \mathbb{R}^n: x \geqq 0\}$ of $\mathbb{R}^n$ and the positive orthant $\{x \in \mathbb{R}^n: x > 0\}$ of $\mathbb{R}^n$, respectively. Let

$$S = \{(x, y) \in \mathbb{R}^{2n}_+: y = Mx + q\},$$

$$S_{\text{int}} = \{(x, y) \in \mathbb{R}^{2n}_{++}: y = Mx + q\}.$$

We assume that $S_{\text{int}} \neq \varnothing$. Define $H: \mathbb{R}^{2n}_+ \times \mathbb{R}_+ \to \mathbb{R}^{2n}$ by

$$H(x, y, \mu) = \begin{bmatrix} XYe - \mu e \\ y - Mx - q \end{bmatrix}, \tag{2.1}$$

where $X = \operatorname{diag}(x_1, \cdots, x_n) \in \mathbb{R}^{n\times n}$ for $x = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$ and $e = (1, \cdots, 1)^T$. Note that the problem (LCP) is equivalent to the system of equations

$$H(x, y, 0) = 0.$$

The mapping H in (2.1) is connected to the following optimization problem with the logarithmic barrier function and $\mu > 0$:

$$\begin{aligned} &\text{minimize} && x^T y - \mu \sum_{i=1}^{n} \ln(x_i y_i), \\ &\text{subject to} && y = Mx + q, \qquad x > 0, \qquad y > 0. \end{aligned} \tag{2.2}$$

Since the objective function in (2.2) is strictly convex, (2.2) is equivalent to the Karush-Kuhn-Tucker conditions

$$\begin{aligned} &y - \mu X^{-1} e + M^T u = 0, \\ &x - \mu Y^{-1} e - u = 0, \\ &y = Mx + q. \end{aligned} \tag{2.3}$$

Multiplying the first equation by X and substracting the second one multiplied by Y, we have

$$X(M^T + X^{-1}Y)u = 0,$$

which implies $u = 0$ since $x > 0$, $y > 0$, and M is positive semidefinite. Thus (2.3) is equivalent to

$$H(x, y, \mu) = 0. \tag{2.4}$$

From the simple fact that for fixed $\mu > 0$, $\zeta - \mu \ln \zeta \to +\infty$ when $\zeta \to +\infty$ we deduce that for each $\mu > 0$ there exists a unique solution $(x(\mu), y(\mu))$ to (2.2). Hence equation (2.4) has a unique solution $(x(\mu), y(\mu))$ for any $\mu > 0$.

The Fréchet partial derivative of H with respect to (x, y) is

$$H_{(x,y)}(x, y, \mu) = \begin{bmatrix} Y & X \\ -M & I \end{bmatrix}. \tag{2.5}$$

Since

$$\begin{bmatrix} I & 0 \\ MY^{-1} & I \end{bmatrix}\begin{bmatrix} Y & X \\ -M & I \end{bmatrix} = \begin{bmatrix} Y & X \\ 0 & I+MY^{-1}X \end{bmatrix},$$

the matrix $H_{(x,y)}(x, y, \mu)$ is nonsingular for all $(x, y) \in \mathbb{R}^{2n}_{++}$ and $\mu > 0$. By using the implicit function theorem repeatedly, we have the following proposition.

PROPOSITION 2.1. *The solution path* $(x(\mu), y(\mu))$ *of* (2.4) *is smooth for* $\mu > 0$.

The algorithm presented here will follow the homotopy curve defined by (2.4) numerically. We use Euler's method as a predictor and follow it by a one-step Newton's iteration. We first introduce a quantity to measure how far away a given point is from the path. For any $(x, y) > 0$, the orthogonal projection of the vector XYe to the straight line $\{\mu e: \mu \in \mathbb{R}\}$ is given by $(x^T y/n)\, e$. Therefore a point (x, y) in S_{int} is on the solution path if and only if $XYe - (x^T y/n)\, e = 0$. This leads to the definition of a tube neighborhood $S(\theta)$ of the path

$$S(\theta) = \left\{(x, y) \in S_{\text{int}}: \left\| XYe - \frac{x^T y}{n} e \right\| \leqq \theta \frac{x^T y}{n}\right\}. \tag{2.6}$$

ALGORITHM. We assume that an initial point $(x^1, y^1) \in S(1/8)$ with $(x^1)^T y^1 \leqq 2^{O(L)}$ is known in advance, where L is the size of the problem (LCP).

Step 0. Let $\delta = (1/2\sqrt{n})$ and $\varepsilon = 2^{-2L}$. Let $k = 1$.

Step 1. If $(x^k)^T y^k < \varepsilon$, then stop. Otherwise go to Step 2.

Step 2. Let $\mu = ((x^k)^T y^k)/n$. Let $(x, y) = (x^k, y^k)$. Compute the Euler direction $(\Delta x, \Delta y)$ by

$$Y\Delta x + X\Delta y = \delta\mu e \quad \text{and} \quad \Delta y = M\Delta x \tag{2.7}$$

and let

$$\tilde{x} = x - \Delta x \quad \text{and} \quad \tilde{y} = y - \Delta y.$$

Step 3. Let $\tilde{\mu} = (1-\delta)\mu$. Compute the Newton direction $(\Delta\tilde{x}, \Delta\tilde{y})$ from

$$\tilde{Y}\Delta\tilde{x} + \tilde{X}\Delta\tilde{y} = \tilde{X}\tilde{Y}e - \tilde{\mu}e \quad \text{and} \quad \Delta\tilde{y} = M\Delta\tilde{x} \tag{2.8}$$

and let

$$\hat{x} = \tilde{x} - \Delta\tilde{x} \quad \text{and} \quad \hat{y} = \tilde{y} - \Delta\tilde{y}.$$

Step 4. Let $(x^{k+1}, y^{k+1}) = (\hat{x}, \hat{y})$ and $k := k+1$. Go to Step 1.

Remark 1. Step 2 is the application of one-step Euler's method to the ordinary differential equations associated with the homotopy equation (2.4), starting from $(x^k, y^k, ((x^k)^T y^k)/n)$ with step-size $\Delta\mu = -\delta(((x^k)^T y^k)/n)$. To see this, we differentiate the identity $H(x(\mu), y(\mu), \mu) = 0$ with respect to μ to obtain

$$H_{(x,y)}(x(\mu), y(\mu), \mu)\begin{bmatrix} \dot{x}(\mu) \\ \dot{y}(\mu) \end{bmatrix} + H_\mu(x(\mu), y(\mu), \mu) = 0. \tag{2.9}$$

By substituting (2.5) and $H_\mu(x, y, \mu) = (-e, 0)^T$ into (2.9), we have

$$\begin{bmatrix} Y & X \\ -M & I \end{bmatrix}\begin{bmatrix} \dot{x}(\mu) \\ \dot{y}(\mu) \end{bmatrix} = \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

Hence

$$\begin{bmatrix} \dot{x}(\mu) \\ \dot{y}(\mu) \end{bmatrix} = \begin{bmatrix} Y & X \\ -M & I \end{bmatrix}^{-1} \begin{bmatrix} e \\ 0 \end{bmatrix}.$$

Now, let $\Delta\mu = -\delta\mu$ with $\mu = ((x^k)^T y^k)/n$ and one Euler's step from (x^k, y^k) gives

$$\begin{aligned} \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} &= \begin{bmatrix} x^k \\ y^k \end{bmatrix} + \Delta\mu \begin{bmatrix} \dot{x}(\mu) \\ \dot{y}(\mu) \end{bmatrix} \\ &= \begin{bmatrix} x^k \\ y^k \end{bmatrix} - \delta\mu \begin{bmatrix} Y & X \\ -M & I \end{bmatrix}^{-1} \begin{bmatrix} e \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} x^k \\ y^k \end{bmatrix} - \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix}, \end{aligned}$$

with $(\Delta x, \Delta y)$ satisfying (2.7).

Remark 2. Step 3 is just the one-step Newton's iteration for (2.4) with $\mu = \tilde{\mu}$ starting from $(\tilde{x}, \tilde{y})$ obtained in Step 2.

Remark 3. For our algorithm, an initial point $(x^1, y^1) \in S(\frac{1}{8})$ is needed in advance. We may use the device given in [9] to construct an artificial linear complementarity problem, having a trivial initial point satisfying all the conditions for our purpose.

3. Convergence theorem of the algorithm. In this section, we give a detailed discussion of our algorithm.

First of all, notice that, when $(x, y) \in S_{\text{int}}$, the new point $(\hat{x}, \hat{y})$ obtained from one-step Euler's prediction followed by one-step Newton's iteration always satisfies $\hat{y} = M\hat{x} + q$. This can be verified easily from (2.7) and (2.8).

Now let $\theta, \delta \in (0, 1)$ be two fixed positive numbers and $(x, y) \in S(\theta)$. Let $\mu = (x^T y)/n$. Consider the following equations:

$$Y\Delta x + X\Delta y = \delta\mu e \quad \text{and} \quad \Delta y = M\Delta x. \tag{3.1}$$

Suppose $(\Delta x, \Delta y)$ is the unique solution. Let $\tilde{x} = x - \Delta x$ and $\tilde{y} = y - \Delta y$. The following lemma is quite useful in our analysis.

LEMMA 3.1 [9]. *If p, r, and u in* $\mathbb{R}^n$ *satisfy*

$$p + r = u, \qquad p^T r \geqq 0,$$

then

(i) $\max\{\|p\|, \|r\|\} \leqq \|u\|$,

(ii) $\|p\|\,\|r\| \leqq \|u\|^2/2$.

Proof. (i) is from the fact that

$$\|u\|^2 = \|p + r\|^2 = \|p\|^2 + \|r\|^2 + 2p^T r \geqq \|p\|^2 + \|r\|^2,$$

and (ii) is true since

$$\|p\|\,\|r\| \leqq (\|p\|^2 + \|r\|^2)/2 \leqq \|u\|^2/2. \qquad \square$$

LEMMA 3.2. *Let* $\tilde{\mu} = (1 - \delta)\mu$. *Then*

$$\|\tilde{X}\tilde{Y}e - \tilde{\mu}e\| \leqq \left[\theta + \frac{n\delta^2}{2(1-\theta)}\right](1-\delta)^{-1}\tilde{\mu}. \tag{3.2}$$

Proof. Multiply both sides of the first equation in (3.1) by $(XY)^{-1/2}$ and denote $(XY^{-1})^{1/2}$ by D; we have

$$D^{-1}\Delta x + D\Delta y = \delta\mu(XY)^{-1/2} e.$$

Let $u = \delta\mu(XY)^{-1/2} e$. Since $(x, y) \in S(\theta)$, we have

$$(1-\theta)\mu \leqq x_i y_i \leqq (1+\theta)\mu, \qquad i = 1, \cdots, n.$$

So, from the definition of the spectral norm of a matrix,

$$\|(XY)^{-1/2}\| = \max_i |x_i y_i|^{-1/2} \leqq (1-\theta)^{-1/2} \cdot \mu^{-1/2}.$$

Thus,

$$\begin{aligned}\|u\| &= \|\delta\mu(XY)^{-1/2} e\| \leqq \delta\mu(1-\theta)^{-1/2} \cdot \mu^{-1/2} \cdot \sqrt{n} \\ &= \sqrt{n}\,(1-\theta)^{-1/2}\mu^{1/2}\delta.\end{aligned}$$

Since M is positive semidefinite, we have

$$(D^{-1}\Delta x)^T (D\Delta y) = \Delta x^T \Delta y = \Delta x^T M \Delta x \geqq 0.$$

By Lemma 3.1, $\|D^{-1}\Delta x\| \leqq \|u\|$, $\|D\Delta y\| \leqq \|u\|$, and $\|D^{-1}\Delta x\|\,\|D\Delta y\| \leqq \|u\|^2/2$. Hence,

$$\begin{aligned}\|\Delta X \Delta y\| &= \|D^{-1}\Delta X D \Delta y\| \leqq \|D^{-1}\Delta X\|\,\|D\Delta y\| \\ &\leqq \|D^{-1}\Delta x\|\,\|D\Delta y\| \leqq \|u\|^2/2 \leqq \frac{n\mu\delta^2}{2(1-\theta)}.\end{aligned}$$

Now

$$\begin{aligned}\tilde{X}\tilde{Y}e &= (X - \Delta X)(Y - \Delta Y)\, e \\ &= XYe - (Y\Delta x + X\Delta y) + \Delta X \Delta y \\ &= XYe - \delta\mu\, e + \Delta X \Delta y.\end{aligned}$$

Let $\tilde{\mu} = (1-\delta)\mu$. Then

$$\begin{aligned}\|\tilde{X}\tilde{Y}e - \tilde{\mu}e\| &= \|XYe - \tilde{\mu}e - \delta\mu e + \Delta X \Delta y\| \\ &\leqq \|XYe - \mu\, e\| + \|\Delta X \Delta y\| \\ &\leqq \theta\mu + \frac{n\mu\delta^2}{2(1-\theta)} = \left[\theta + \frac{n\delta^2}{2(1-\theta)}\right]\mu \\ &= \left[\theta + \frac{n\delta^2}{2(1-\theta)}\right](1-\delta)^{-1}\tilde{\mu}.\end{aligned}$$

□

Let $\tilde{\theta} = [\theta + (n\delta^2/2(1-\theta))](1-\delta)^{-1}$. We have the following proposition.

PROPOSITION 3.3. *If* $\tilde{\theta} < 1$, *then* $(\tilde{x}, \tilde{y}) > 0$.

Proof. Since $\|\tilde{X}\tilde{Y}e - \tilde{\mu}\, e\| \leqq \tilde{\theta}\tilde{\mu} < \tilde{\mu}$, we have for $i = 1, \cdots, n$, $|\tilde{x}_i\tilde{y}_i - \tilde{\mu}| < \tilde{\mu}$, so $\tilde{x}_i\tilde{y}_i > 0$. If for some i, $\tilde{x}_i < 0$ and $\tilde{y}_i < 0$, then $0 < x_i < \Delta x_i$ and $0 < y_i < \Delta y_i$. Hence, $0 < x_i y_i < \Delta x_i \Delta y_i \leqq \|\Delta X \Delta y\| \leqq n\mu\delta^2/2(1-\theta)$. On the other hand, $x_i y_i \geqq (1-\theta)\mu$ since $(x, y) \in S(\theta)$. Thus, $(1-\theta)\mu < n\mu\delta^2/2(1-\theta)$, i.e., $1 < \theta + (n\delta^2/2(1-\theta))$. But $\theta + (n\delta^2/2(1-\theta)) < (\theta/(1-\delta)) + (n\delta^2/2(1-\theta)(1-\delta)) = \tilde{\theta} < 1$. This is a contradiction. Therefore $(\tilde{x}, \tilde{y}) > 0$. □

Now we turn to the following equations for Newton's iteration:

$$\tilde{Y}\Delta\tilde{x} + \tilde{X}\Delta\tilde{y} = \tilde{X}\tilde{Y}e - \tilde{\mu}\, e, \qquad \Delta\tilde{y} = M\Delta\tilde{x}, \tag{3.3}$$

and define $\hat{x}=\tilde{x}-\Delta\tilde{x}$ and $\hat{y}=\tilde{y}-\Delta\tilde{y}$, where $(\Delta\tilde{x},\Delta\tilde{y})$ is the unique solution of (3.3) when $\tilde{\theta}<1$. Let $\hat{\mu}=(\hat{x}^T\hat{y})/n$.

LEMMA 3.4. *Let* $\hat{\theta}=\tilde{\theta}^2/2(1-\tilde{\theta})$. *If* $\tilde{\theta}<1$, *then*

(i) $\|\hat{X}\hat{Y}e-\hat{\mu}e\|\leqq\hat{\theta}\hat{\mu}$,

(ii) $\hat{\mu}\leqq(1-\delta)(1+(\hat{\theta}/\sqrt{n}))\mu$.

Proof. Multiply both sides of the first equation in (3.3) by $(\tilde{X}\tilde{Y})^{-1/2}$ and denote $(\tilde{X}\tilde{Y}^{-1})^{1/2}$ by $\tilde{D}$, we have

$$\tilde{D}^{-1}\Delta\tilde{x}+\tilde{D}\Delta\tilde{y}=(\tilde{X}\tilde{Y})^{-1/2}(\tilde{X}\tilde{Y}e-\tilde{\mu}e)\equiv\tilde{u}.$$

Since $\|\tilde{X}\tilde{Y}e-\tilde{\mu}e\|\leqq\tilde{\theta}\tilde{\mu}$, we have $(1-\tilde{\theta})\tilde{\mu}\leqq\tilde{x}_i\tilde{y}_i\leqq(1+\tilde{\theta})\tilde{\mu}$ for $i=1,\cdots,n$. Hence,

$$\|(\tilde{X}\tilde{Y})^{-1/2}\|=\max_i|\tilde{x}_i\tilde{y}_i|^{-1/2}\leqq(1-\tilde{\theta})^{-1/2}\tilde{\mu}^{-1/2}.$$

Thus,

$$\begin{aligned}\|\tilde{u}\|&\leqq\|(\tilde{X}\tilde{Y})^{-1/2}\|\,\|\tilde{X}\tilde{Y}e-\tilde{\mu}e\|\leqq(1-\tilde{\theta})^{-1/2}\cdot\tilde{\mu}^{-1/2}\cdot\tilde{\theta}\cdot\tilde{\mu}\\&=\tilde{\theta}(1-\tilde{\theta})^{-1/2}\tilde{\mu}^{1/2}.\end{aligned}$$

Furthermore, from $(\tilde{D}^{-1}\Delta\tilde{x})^T(\tilde{D}\Delta\tilde{y})=\Delta\tilde{x}^TM\Delta\tilde{x}\geqq0$ and Lemma 3.1, we obtain $\|\tilde{D}^{-1}\Delta\tilde{x}\|\leqq\|\tilde{u}\|$, $\|\tilde{D}\Delta\tilde{y}\|\leqq\|\tilde{u}\|$, and $\|\tilde{D}^{-1}\Delta\tilde{x}\|\,\|\tilde{D}\Delta\tilde{y}\|\leqq\|\tilde{u}\|^2/2$. Thus,

$$\|\Delta\tilde{X}\Delta\tilde{y}\|\leqq\|\tilde{D}^{-1}\Delta\tilde{x}\|\,\|\tilde{D}\Delta\tilde{y}\|\leqq\|\tilde{u}\|^2/2\leqq\frac{\tilde{\theta}^2\tilde{\mu}}{2(1-\tilde{\theta})}.$$

Now

$$\begin{aligned}\hat{X}\hat{Y}e&=(\tilde{X}-\Delta\tilde{X})(\tilde{Y}-\Delta\tilde{Y})\,e\\&=\tilde{X}\tilde{Y}e-[\tilde{Y}\Delta\tilde{x}+\tilde{X}\Delta\tilde{y}]+\Delta\tilde{X}\Delta\tilde{y}\\&=\tilde{X}\tilde{Y}e-(\tilde{X}\tilde{Y}e-\tilde{\mu}e)+\Delta\tilde{X}\Delta\tilde{y}\\&=\tilde{\mu}e+\Delta\tilde{X}\Delta\tilde{y}.\end{aligned}$$

Hence,

$$\|\hat{X}\hat{Y}e-\tilde{\mu}e\|=\|\Delta\tilde{X}\Delta\tilde{y}\|\leqq\frac{\tilde{\theta}^2\tilde{\mu}}{2(1-\tilde{\theta})}. \tag{3.4}$$

Put $\hat{\mu}=(\hat{x}^T\hat{y})/n$, then $\hat{\mu}e$ is the orthogonal projection of the vector $\hat{X}\hat{Y}e$ to the diagonal $\{\lambda e:\lambda\in\mathbb{R}\}$. This means that $\|\hat{X}\hat{Y}e-\hat{\mu}e\|=\min_{\lambda\in\mathbb{R}}\|\hat{X}\hat{Y}e-\lambda e\|\leqq\|\hat{X}\hat{Y}e-\tilde{\mu}e\|$. On the other hand, we calculate the inner product

$$\begin{aligned}\hat{x}^T\hat{y}&=e^T\hat{X}\hat{Y}e\\&=\tilde{\mu}e^Te+e^T\Delta\tilde{X}\Delta\tilde{y}\\&=n\tilde{\mu}+\Delta\tilde{x}^T\Delta\tilde{y}.\end{aligned} \tag{3.5}$$

Since $\Delta\tilde{x}^T\Delta\tilde{y}=\Delta\tilde{x}^TM\Delta\tilde{x}\geqq0$, we have $\hat{x}^T\hat{y}\geqq n\tilde{\mu}$ and hence $\hat{\mu}\geqq\tilde{\mu}$. The relation $\mu\geqq\hat{\mu}\geqq\tilde{\mu}$ means that the Euler step gives a "too large" decrease of μ, that we are relatively far from the trajectory, and that the Newton step corrects this drawback. Substituting into (3.4), we have

$$\|\hat{X}\hat{Y}e-\hat{\mu}e\|\leqq\frac{\tilde{\theta}^2}{2(1-\tilde{\theta})}\hat{\mu}=\hat{\theta}\hat{\mu}.$$

This proves (i). Again, from (3.5), we obtain

$$\hat{\mu} = \frac{\hat{x}^T\hat{y}}{n} = \tilde{\mu} + \frac{e^T}{n}\Delta\tilde{X}\Delta\tilde{y} \leqq \tilde{\mu} + \frac{\tilde{\theta}^2\tilde{\mu}}{2\sqrt{n}\,(1-\tilde{\theta})}$$
$$= \left[1 + \frac{\tilde{\theta}^2}{2\sqrt{n}\,(1-\tilde{\theta})}\right]\tilde{\mu} = (1-\delta)\left[1 + \frac{\tilde{\theta}^2}{2\sqrt{n}\,(1-\tilde{\theta})}\right]\mu$$
$$= (1-\delta)\left(1 + \frac{\hat{\theta}}{\sqrt{n}}\right)\mu,$$

which is (ii). □

PROPOSITION 3.5. *If* $\tilde{\theta} \leqq 2-\sqrt{2}$, *then*

(i) $\|\hat{X}\hat{Y}e - \hat{\mu}e\| \leqq \hat{\theta}\hat{\mu}$,

(ii) $\hat{\mu} \leqq (1-\delta)(1+(\hat{\theta}/\sqrt{n}))\mu$,

(iii) $(\hat{x}, \hat{y}) \in S(\hat{\theta})$ *with* $\hat{\theta} \leqq (3-2\sqrt{2})/(\sqrt{2}-1)$.

Proof. Since $2-\sqrt{2} < 1$, from Lemma 3.4 only $\hat{x} > 0$, $\hat{y} > 0$ is to be proved. Suppose $\tilde{\theta} \leqq 2-\sqrt{2}$. From Proposition 3.3, $(\tilde{x}, \tilde{y}) > 0$. Thus $(\hat{x}, \hat{y})$ is well defined. Since $\hat{\theta} = \tilde{\theta}^2/2(1-\tilde{\theta})$ is an increasing function of $\tilde{\theta} \in (0, 1)$, $\tilde{\theta} \leqq 2-\sqrt{2}$ implies that $\hat{\theta} \leqq (2-\sqrt{2})^2/[2(1-2+\sqrt{2})] = (3-2\sqrt{2})/(\sqrt{2}-1) < 1$. It follows from (i) that $\|\hat{X}\hat{Y}e - \hat{\mu}e\| < \hat{\mu}$. Thus $\hat{x}_i\hat{y}_i > 0$ for $i = 1, \cdots, n$. If for some i both $\hat{x}_i < 0$ and $\hat{y}_i < 0$, then $0 < \tilde{x}_i < \Delta\tilde{x}_i$ and $0 < \tilde{y}_i < \Delta\tilde{y}_i$. This leads to

$$(1-\tilde{\theta})\tilde{\mu} \leqq \tilde{x}_i\tilde{y}_i < \Delta\tilde{x}_i\Delta\tilde{y}_i \leqq \|\Delta\tilde{X}\Delta\tilde{y}\| \leqq \frac{\tilde{\theta}^2}{2(1-\tilde{\theta})}\tilde{\mu}.$$

So $(1-\tilde{\theta})^2 < \tilde{\theta}^2/2$, i.e.,

$$\tfrac{1}{2}\tilde{\theta}^2 - 2\tilde{\theta} + 1 < 0,$$

which implies that $\tilde{\theta} > 2-\sqrt{2}$. Therefore $(\hat{x}, \hat{y}) > 0$ and thus by (i), $(\hat{x}, \hat{y}) \in S(\hat{\theta})$.

As an easy consequence of the above preliminary results, we come to the following theorem.

THEOREM 3.6. *Suppose* $(x, y) \in S(\theta)$. *Let* $\tilde{\theta} = [\theta + (n\delta^2/2(1-\theta))](1-\delta)^{-1}$ *and* $\hat{\theta} = \tilde{\theta}^2/2(1-\tilde{\theta})$. *If we choose the parameters* $\theta > 0$ *and* $\delta > 0$ *such that*

$$\tilde{\theta} \leqq 2-\sqrt{2} \quad \textit{and} \quad \hat{\theta} \leqq \theta,$$

then the new point $(\hat{x}, \hat{y}) = (\tilde{x} - \Delta\tilde{x}, \tilde{y} - \Delta\tilde{y}) = (x - \Delta x - \Delta\tilde{x}, y - \Delta y - \Delta\tilde{y})$ *with* $(\Delta x, \Delta y)$ *and* $(\Delta\tilde{x}, \Delta\tilde{y})$ *satisfying the equations* (3.1) *and* (3.3), *respectively, belongs to* $S(\theta)$, *and satisfies*

$$\hat{x}^T\hat{y} \leqq (1-\delta)\left[1 + \frac{\hat{\theta}}{\sqrt{n}}\right]x^Ty. \tag{3.6}$$

In particular, if we let $\theta = \frac{1}{8}$ *and* $\delta = 1/2\sqrt{n}$, *then the corresponding* $(\hat{x}, \hat{y}) \in S(\theta)$ *and*

$$\hat{x}^T\hat{y} \leqq \left(1 - \frac{2}{5\sqrt{n}}\right)x^Ty. \tag{3.7}$$

Proof. It should be noted that $\hat{\theta} \leqq \theta$ implies $S(\hat{\theta}) \subset S(\theta)$, and formulas (3.6) and (3.7) hold for $\hat{\mu}$ as well. We only need to check the last statement. If $\theta = \frac{1}{8}$ and $\delta = 1/2\sqrt{n}$, then

$$\tilde{\theta} = \left[\theta + \frac{n\delta^2}{2(1-\theta)}\right](1-\delta)^{-1} = \left[\frac{1}{8} + \frac{1}{8\cdot(1-\frac{1}{8})}\right]\frac{2}{2-1/\sqrt{n}}$$
$$= \left[\frac{1}{8} + \frac{1}{7}\right]\frac{2}{2-1/\sqrt{n}} = \frac{15}{28}\frac{1}{2-1/\sqrt{n}} < 2-\sqrt{2}.$$

Taking account of our assumption that $n \geqq 3$, we get

$$\begin{aligned}\hat{\theta} &= \frac{\tilde{\theta}^2}{2(1-\tilde{\theta})} \\ &= \left(\frac{15}{28}\right)^2 \Big/ \left[2\left(1-\frac{15}{28(2-1/\sqrt{n})}\right)\left(2-\frac{1}{\sqrt{n}}\right)^2\right] \\ &= 225 \Big/ \left[2\cdot 28\left(41-\frac{28}{\sqrt{n}}\right)\left(2-\frac{1}{\sqrt{n}}\right)\right] \\ &\leqq 3\cdot 225/[56\cdot(41\sqrt{3}-28)(2\sqrt{3}-1)] \\ &< 0.114 < \frac{1}{8} = \theta;\end{aligned}$$

and finally

$$\begin{aligned}\hat{x}^T\hat{y} &\leqq (1-\delta)\left[1+\frac{\hat{\theta}}{\sqrt{n}}\right]x^Ty = \left[1-\delta+\frac{(1-\delta)\hat{\theta}}{\sqrt{n}}\right]x^Ty \\ &= \left[1-\frac{1}{2\sqrt{n}}+\frac{225}{2\cdot 28\cdot(41-28/\sqrt{n})\cdot 2\sqrt{n}}\right]x^Ty \\ &\leqq \left[1-\left(\frac{1}{2}-\frac{225}{112\cdot(41-28/\sqrt{3})}\right)\frac{1}{\sqrt{n}}\right]x^Ty \\ &< \left(1-\frac{2}{5\sqrt{n}}\right)x^Ty. \qquad \square\end{aligned}$$

Theorem 3.6 constitutes the basis for our algorithm in § 2. From a known initial point $(x^1, y^1) \in S(\frac{1}{8})$, and choosing $\delta = 1/2\sqrt{n}$, the sequence $\{(x^k, y^k)\}$ generated by the algorithm lies in the $\frac{1}{8}$-neighborhood $S(\frac{1}{8})$ of the homotopy path $(x(\mu), y(\mu))$ and the values $(x^k)^Ty^k$ decrease at least linearly with the global convergence ratio $(1-(2/5\sqrt{n}))$ along the sequence. More specifically, for each k, we have

$$(x^{k+1})^Ty^{k+1} < \left(1-\frac{2}{5\sqrt{n}}\right)^k (x^1)^Ty^1, \tag{3.8}$$

and thus we can prove the following convergence result.

COROLLARY 3.7. *In* $\lceil \frac{5}{2}(\ln 2)\sqrt{n}\cdot t \rceil$ *steps the algorithm finds a feasible point* $(x, y) \in S_{\text{int}}$ *such that*

$$\frac{x^Ty}{(x^1)^Ty^1} < 2^{-t},$$

where $\lceil a \rceil$ *denotes the smallest integer greater than or equal to* a.

Proof. From (3.8), we need to solve the inequality

$$\left(1-\frac{2}{5\sqrt{n}}\right)^k < 2^{-t}.$$

The solution is

$$k \geqq -t/\log_2\left(1-\frac{2}{5\sqrt{n}}\right) = \ln 2\cdot t \Big/ \left[-\ln\left(1-\frac{2}{5\sqrt{n}}\right)\right].$$

Since $-\ln(1-(2/5\sqrt{n})) > 2/5\sqrt{n}$, the assertion is achieved. $\square$

It is obvious from the above corollary that if the initial point (x^1, y^1) satisfies $(x^1)^T y^1 \leqq 2^{O(L)}$, then after $O(\sqrt{n}\, L)$ iterations, the algorithm finds an approximate solution $(\hat{x}, \hat{y}) = (x^k, y^k)$ of the problem (LCP) satisfying

$$(\hat{x}, \hat{y}) \in S \quad \text{and} \quad \hat{x}^T \hat{y} < 2^{-2L}.$$

Here L is the size of the problem.

At each iteration cycle, our algorithm solves two linear equations (2.7) and (2.8) successively. Thus, it requires around $2n^3$ arithmetic operations. The $O(n^{3.5}L)$ algorithm presented by Kojima, Mizuno, and Yoshise has decreasing rate $(1-(1/8\sqrt{n}))$ with n^3 arithmetic operations at each iteration. Then, after every two iterations in their method, the value $x^T y$ will decrease at the rate of at least $(1-(1/8\sqrt{n}))^2 = (1-(1/4\sqrt{n})+(1/64n))$ with $2n^3$ arithmetic operations, while for our method the value $x^T y$ will decrease at the rate at least $(1-(2/5\sqrt{n}))$ with the same number of arithmetic operations.

4. Conclusions. In this paper, we presented a polynomial-time predictor-corrector algorithm for the (LCP). With Euler's method as first-order predictor followed by one-step Newton's method as corrector, this method is shown to be an improvement in the efficiency of the basic algorithm in [9] with zero-order predictor. This different type of approach seems more natural from both the general viewpoint of the homotopy continuation method and the special form of the homotopy equation (2.1), which is actually a quadratic polynomial system. Our preliminary numerical experiments show that this approach appears quite efficient for the (LCP).

As an alternative, we may use the linear interpolation technique in place of the Euler's scheme in our algorithm to avoid solving linear equation (2.7), and thus only about half of the numerical work is needed.

A common shortage of the current interior point methods is the requirement of an interior point $x_0 > 0$ and $y_0 > 0$ satisfying (2.6). As a partial solution to this problem, the "weighted" homotopy path may be followed from any initial strictly feasible point, as is described in [3]. The elimination of the "Phase 1" problem as well as the numerical efficiency investigation of the homotopy method for the (LCP) are still major research focuses in this area.

REFERENCES

[1] E. L. ALLGOWER AND K. GEORG, *Predictor-corrector and simplicial methods for approximating fixed points and zero points of nonlinear mappings*, in Mathematical Programming: The State of the Art, A. Bacham, M. Grotschel, and B. Korte, eds., Springer-Verlag, New York, 1983.

[2] M. BEN DAYA AND C. M. SHETTY, *Polynomial barrier function algorithms for convex quadratic programming*, Res. Report No. J 88-5, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 1988.

[3] J. DING AND T. Y. LI, *An algorithm based on weighted logarithmic barrier functions for linear complementarity problems*, to appear in the Theme Issue Optimization: Theory and Engineering Applications, Arabian J. Sciences and Engineering, 15, 4(B) (1990), pp. 679-685.

[4] A. FIACCO AND G. MCCORMICK, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley, New York, 1968.

[5] K. R. FRISH, *The logarithmic potential method of convex programming*, Memorandum, University Institute of Economics, Oslo, Norway, 1955.

[6] C. C. GONZAGA, *Large-steps path-following methods for linear programming: Barrier function method*, Report ES-2/0/89, Department of Systems Engineering and Computer Sciences, COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, 1989.

[7] ———, *Large-steps path-following methods for linear programming: Potential reduction method*, Report ES-211/89, Department of Systems Engineering and Computer Sciences, COPPE-Federal University of Rio de Janeiro, Rio de Janeiro, Brazil, 1989.

[8] N. KARMARKAR, *A new polynomial-time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373-395.

[9] M. KOJIMA, S. MIZUNO, AND A. YOSHISE, *A polynomial-time algorithm for a class of linear complementarity problems*, Math. Programming, 44 (1989), pp. 1-26.

[10] ———, *An $O(\sqrt{n}\,L)$ potential reduction algorithm for linear complementarity problems*, Res. Report, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1987.

[11] ———, *A primal-dual interior point method for linear programming*, Res. Report B-188, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1987.

[12] N. MEGIDDO, *Pathways to the optimal set in linear programming*, in Proc. 6th Mathematical Programming Symposium of Japan, Nagoya, Japan, 1986, pp. 1-35.

[13] R. C. MONTEIRO AND I. ADLER, *Interior path-following primal-dual algorithms, Part* I: *Linear programming*, Math. Programming, 44 (1989), pp. 27-41.

[14] ———, *Interior path-following primal-dual algorithms, Part* II: *Convex quadratic programming*, Math. Programming, 44 (1989), pp. 43-66.

[15] K. G. MURTY, *Linear Complementarity, Linear and Nonlinear Programming*, Heldermann-Verlag, Berlin, 1988.

[16] J. RENEGAR, *A polynomial-time algorithm based on Newton's method for linear programming*, Math. Programming, 40 (1988), pp. 59-94.

[17] J. RENEGAR AND M. SHUB, *Simplified complexity analysis for Newton* LP *methods*, Tech. Report 807, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1988.

[18] Y. YE, *Interior algorithms for linear, quadratic and linearly constrained convex programming*, Ph.D. thesis, Department of Engineering-Economic Systems, Stanford University, Stanford, CA, 1987.

[19] ———, *Further development on the interior algorithms for convex quadratic programming*, preprint, Department of Engineering-Economic Systems, Stanford University, Stanford, CA, 1987.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 93–113, February 1991

ON THE SOLUTION OF LARGE QUADRATIC PROGRAMMING PROBLEMS WITH BOUND CONSTRAINTS*

JORGE J. MORÉ† AND GERARDO TORALDO‡

Abstract. An algorithm is proposed that uses the conjugate gradient method to explore the face of the feasible region defined by the current iterate, and the gradient projection method to move to a different face. It is proved that for strictly convex problems the algorithm converges to the solution, and that if the solution is nondegenerate, then the algorithm terminates at the solution in a finite number of steps. Numerical results are presented for the obstacle problem, the elastic-plastic torsion problem, and the journal bearing problems. On a selection of these problems with dimensions ranging from 5000 to 15,000, the algorithm determines the solution in fewer than 15 iterations, and with a small number of function-gradient evaluations and Hessian-vector products per iteration.

Key words. quadratic programming, large-scale, conjugate gradients, gradient projection

AMS(MOS) subject classifications. 65K10, 90C20

1. Introduction.

Given a quadratic function $q : \mathbb{R}^n \to \mathbb{R}$, and vectors l and u that specify bounds on the variables, the bound constrained quadratic programming problem is to find a vector x that solves the problem

$$\min \{q(x) : l \le x \le u\}. \tag{1.1}$$

Our concern in this paper is with the solution of this problem when the quadratic q is strictly convex and the number of variables n is large.

The large-scale bound constrained quadratic programming problem has attracted much attention. Recent works on this problem include Cottle, Golub, and Sacher [12]; Cottle and Goheen [11]; O'Leary [24]; Dembo and Tulowitzki [13]; Lin and Cryer [19]; Lin and Pang [20]; Coleman and Hulbert [9]; Björck [3]; Yang and Tolle [27]; and Júdice and Pires [17], [18].

Applications which lead to bound constrained quadratic programming problems include contact and friction problems in rigid body mechanics (Lötstedt [21]), journal bearing lubrication (Cimatti [8], Lin and Cryer [19]), flow through a porous medium (Lin and Cryer [19]), and elastic-plastic torsion problems (Glowinski [16, pp. 158–162]). We use finite element approximations to the elastic-plastic torsion problem and the journal bearing lubrication problem for our numerical results.

Standard algorithms for the solution of (1.1) usually generate a sequence $\{x_k\}$ that terminates at a solution of (1.1) in a finite number of iterations. Finite termination is typically achieved by solving a sequence of subproblems of the form

$$\min \{q(x_k + d) : d_i = 0,\ i \in \mathcal{W}_k\} \tag{1.2}$$

for some index set $\mathcal{W}_k$, and by restricting the change in $\mathcal{W}_k$ by only dropping or adding one constraint at each iteration. In the standard algorithms the constraints in the set $\mathcal{W}_k$ are a subset of the active constraints chosen so that the solution of (1.2) is a feasible direction.

* Received by the editors September 5, 1989; accepted for publication (in revised form) May 22, 1990. This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, contract W-31-109-Eng-38.

† Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois 60439.

‡ Dipartimento di Matematica, Università della Basilicata, Via N. Sauro 85, 85100, Potenza, Italy.

There are two disadvantages to this type of algorithm when applied to large problems. One of the disadvantages is that the solution of subproblem (1.2) requires the solution of a system of linear equations of size m_k where m_k is the number of constraints not in $\mathcal{W}_k$. This can be expensive if m_k is large relative to n. Another disadvantage is that by only dropping or adding one constraint from $\mathcal{W}_k$ at each iteration, we impose a lower bound on the number of iterations required for convergence. In particular, if there are k_0 constraints active at the initial $\mathcal{W}_0$, but k_s constraints active at the solution, then at least $|k_s - k_0|$ iterations are required for convergence. This can be a serious disadvantage in large problems.

The need for an algorithm that accepts approximate solutions of (1.2) has been recognized for quite a while. For example, O'Leary [24] developed an algorithm, based on the work of Polyak [25], which uses the conjugate gradient method for the solution of subproblem (1.2). In the algorithm analyzed by O'Leary the conjugate gradient method was used to solve (1.2) until either a constraint was violated or (1.2) was solved. If a constraint was violated, it was added to $\mathcal{W}_k$, and the process was repeated. Although the theory required the accurate solution of (1.2), O'Leary noted that this requirement essentially doubled the number of iterations, and thus the numerical results were obtained with an algorithm in which the accuracy required of the conjugate gradient method was refined during the course of the iteration. A related algorithm was proposed by Mittlemann [22] for the solution of optimization problems that arise as discretizations of elliptic variational inequalities with bound constraints.

Dembo and Tulowitzki [13] were the first to propose algorithms that do not require the accurate solution of subproblem (1.2), and are able to drop and add many constraints from $\mathcal{W}_k$ at each iteration. Unfortunately, their convergence results were incomplete.

Yang and Tolle [27] followed the ideas of Dembo and Tulowitzki by proposing algorithms that are able to drop and add many constraints at each iteration. An advantage of the algorithms of Yang and Tolle is that they can be shown to terminate at the solution in a finite number of steps, even for degenerate problems. On the other hand, their algorithms are partially based on the work of Polyak [25], and are thus likely to suffer from the same disadvantages.

Wright [26] proposed a modification of the CGP algorithm of Dembo and Tulowitzki [13], and established convergence results for the modified algorithm. In this type of algorithm the gradient projection method is used until a suitable set $\mathcal{W}_k$ is identified, and then the conjugate gradient method is used to obtain an approximate solution to (1.2). Our experience has been that this strategy can be inefficient because for starting points far from the solution, the gradient projection method can require a large number of iterations before identifying a suitable $\mathcal{W}_k$. A preferable strategy is to use the conjugate gradient method once the gradient projection method fails to make reasonable progress.

The algorithm that we propose in §3 for the solution of problem (1.1) uses the gradient projection method until either a suitable $\mathcal{W}_k$ is identified, or the gradient projection method fails to make reasonable progress. The conjugate gradient method is used to obtain an approximate solution of (1.2) on the current $\mathcal{W}_k$.

The analysis in §5 shows that for strictly convex problems the algorithm converges to the solution of (1.1), and that if the problem is nondegenerate, then the algorithm terminates in a finite number of steps.

Section 6 describes three model problems which arise in applications: the ob-

stacle problem, the elastic-plastic torsion problem, and the journal bearing problem. Section 7 contains results obtained on these model problems with dimensions ranging from 5000 to 15,000. The salient feature of our numerical results is that on these problems the algorithm determines the solution in fewer than 15 iterations. This number is of interest because it represents the number of sets $\mathcal{W}_k$ that are searched before the optimal set is found. The total amount of work required by the algorithm is measured by the number of function-gradient evaluations and Hessian-vector products. In our numerical results the number of function-gradient evaluations per iteration is between 3.8 and 5.1, and the number of Hessian-vector products per iteration is between 10.0 and 41.5.

2. Preliminaries. Problem (1.1) has a unique solution x^* whenever $q : \mathbb{R}^n \to \mathbb{R}$ is strictly convex on the feasible region

$$\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\}. \tag{2.1}$$

This classical result holds for unbounded Ω, and thus we allow the components of the vectors l and u to be infinite. The solution x^* can be characterized as the solution of the variational inequality

$$\langle \nabla q(x^*), x - x^* \rangle \geq 0, \qquad x \in \Omega, \tag{2.2}$$

where $\langle \cdot, \cdot \rangle$ is an inner product and ∇q is the gradient of q with respect to the inner product. Unless otherwise stated, we assume that $\langle \cdot, \cdot \rangle$ is the standard l_2 inner product.

If Ω is the bound constrained set (2.1), any solution of the variational inequality (2.2) satisfies the Kuhn–Tucker conditions

$$\begin{array}{ll} \partial_i q(x) = 0 & \text{if } x_i \in (l_i, u_i) \\ \partial_i q(x) \geq 0 & \text{if } x_i = l_i \\ \partial_i q(x) \leq 0 & \text{if } x_i = u_i \end{array}$$

for a minimizer of problem (1.1). Approximate solutions to problem (1.1) can be defined in terms of the projected gradient defined by

$$[\nabla_\Omega q(x)]_i = \begin{cases} \partial_i q(x) & \text{if } x_i \in (l_i, u_i) \\ \min\{\partial_i q(x), 0\} & \text{if } x_i = l_i \\ \max\{\partial_i q(x), 0\} & \text{if } x_i = u_i. \end{cases} \tag{2.3}$$

This definition of a projected gradient is appropriate for the bound constrained set (2.1). This projected gradient is the negative of the projected gradient defined by Calamai and Moré [5] for a general closed convex set Ω.

Given $x_0 \in \Omega$ and a tolerance τ in $(0, 1)$, we define an approximate solution of problem (1.1) as any vector $x \in \Omega$ such that

$$\|\nabla_\Omega q(x)\| \leq \tau \|\nabla q(x_0)\|. \tag{2.4}$$

The definition (2.3) of the projected gradient shows that $\nabla_\Omega q(x^*) = 0$ whenever x^* is a solution of problem (1.1). Also note that (2.4) holds whenever x is sufficiently close to x^* and in the face of Ω that contains x^*. The concept of a face of a convex set is standard in convex analysis; for the convex set (2.1), the *face* of Ω that contains x is

$$\{y \in \Omega : y_i = x_i \text{ if } x_i \in \{l_i, u_i\}\}.$$

An alternate definition of an approximate minimizer is to require that

$$\|P(x - \nabla q(x)) - x\| \leq \tau \|\nabla q(x_0)\| \tag{2.5}$$

where P is the projection into the feasible region (2.1), that is, $P(x)$ is the closest point in Ω to x. The use of $\|P(x - \nabla q(x)) - x\|$ as a measure of optimality is fairly common. See, for example, Conn, Gould, and Toint [10]. An advantage of (2.4) over (2.5) is that if we replace q by αq for some $\alpha > 0$, then only (2.4) is invariant under this change of scale. Indeed, if we replace q by αq in (2.5) and Ω is bounded, then (2.5) is satisfied by any $x \in \Omega$ if α is sufficiently large.

The projection P into the feasible set (2.1) plays an important role in the algorithms for problem (1.1). We close this section by noting that for the feasible set (2.1) the projection can be computed in order n operations by letting

$$P(x) = \text{mid}(l, u, x), \tag{2.6}$$

where $\text{mid}(l, u, x)$ is the vector whose ith component is the median of the set $\{l_i, u_i, x_i\}$.

3. A bound constrained quadratic programming algorithm. We propose an algorithm for the bound constrained problem (1.1) which uses the conjugate gradient method to explore the face of the feasible region defined by the current iterate and uses the gradient projection method to move to a different face. We first describe the use of the conjugate gradient method.

The face of the feasible set which contains the current iterate can be defined in terms of the active set, where the *active set* $\mathcal{A}(x)$ is defined by

$$\mathcal{A}(x) = \{i : x_i = l_i \text{ or } x_i = u_i\}.$$

The variables with indices in $\mathcal{A}(x)$ are the *active* or *bound* variables, while those with indices outside of $\mathcal{A}(x)$ are the *free* variables. Given the current iterate x_k and the active set $\mathcal{A}(x_k)$, the conjugate gradient method is used to compute an approximate minimizer of the subproblem

$$\min\{q(x_k + d) : d_i = 0,\ i \in \mathcal{A}(x_k)\}. \tag{3.1}$$

This is an unconstrained quadratic programming problem in the free variables. Note that if x_k lies in the same face as the solution of (1.1) and d_k solves (3.1), then $x_k + d_k$ is the solution of (1.1).

The conjugate gradient algorithm for the solution of subproblem (3.1) is implemented by expressing this subproblem in terms of an equivalent subproblem in the free variables. If $i_1, \cdots, i_{m_k}$ are the indices of the free variables, and the matrix Z_k is defined as the matrix in $\mathbb{R}^{n \times m_k}$ whose jth column is the i_jth column of the identity matrix in $\mathbb{R}^{n \times n}$, then subproblem (3.1) is equivalent to the unconstrained subproblem

$$\min\{q_k(w) : w \in \mathbb{R}^{m_k}\}, \tag{3.2}$$

where

$$q_k(w) \equiv q(x_k + Z_k w) - q(x_k) = \tfrac{1}{2} w^T A_k w + r_k^T w.$$

The matrix A_k and the vector r_k are, respectively, the reduced Hessian matrix of q and reduced gradient of q at x_k with respect to the free variables. If A is the Hessian matrix of the quadratic q then

$$A_k = Z_k^T A Z_k, \qquad r_k = Z_k^T \nabla q(x_k). \tag{3.3}$$

Also note that A_k is the matrix obtained from A by taking those rows and columns whose indices correspond to free variables; similarly, r_k is obtained from $\nabla q(x_k)$ by taking the components whose indices correspond to free variables.

Given a starting point $w_0 \in \mathbb{R}^{m_k}$, the conjugate gradient algorithm generates a sequence of iterates $w_0, w_1, \cdots$ that terminates at a solution of subproblem (3.2) in at most m_k iterations. We use the conjugate gradient algorithm until it generates w_j such that

$$q_k(w_{j-1}) - q_k(w_j) \leq \eta_1 \max\{q_k(w_{l-1}) - q_k(w_l) : 1 \leq l < j\} \tag{3.4}$$

for some fixed constant $\eta_1 > 0$. The approximate solution of subproblem (3.1) is then $d_k = Z_k w_{j_k}$, where j_k is the first index j that satisfies (3.4).

Test (3.4) detects when the conjugate gradient method is not making sufficient progress. One of the advantages of this test is that it is scale invariant, that is, the test is unchanged if it is applied to the function αq for any $\alpha > 0$. Another advantage is that this test is eventually satisfied provided the sequence $\{q_k(w_j)\}$ is monotone decreasing and q_k is bounded below. We could have used other tests, but test (3.4) seems to be appropriate for the conjugate gradient method. For example, the test

$$\|\nabla q_k(w_j)\| \leq \eta_1 \|\nabla q_k(0)\|$$

is often used instead of (3.4). This test is scale invariant. Moreover, this test is eventually satisfied because w_j solves subproblem (3.2) for some $j \leq m_k$. On the other hand, the behavior of the residual norm $\|\nabla q_k(w_j)\|$ is often erratic.

In general we do not set $x_{k+1} = x_k + d_k$ because this may produce an infeasible x_{k+1}. Standard algorithms set $x_{k+1} = x_k + \alpha_k d_k$ where α_k is the minimizer of $q(x_k + \alpha d_k)$ in the interval $[0, \mu_k]$ and

$$\mu_k = \max\{\alpha \geq 0 : l \leq x_k + \alpha d_k \leq u\}. \tag{3.5}$$

However, in this strategy only one constraint is usually added at each iteration, and as noted in the introduction, this can lead to an inefficient algorithm. We use a projected search to define α_k, and then set

$$x_{k+1} = P(x_k + \alpha_k d_k) \tag{3.6}$$

where P is the projection (2.6) into the feasible region (2.1). The projected search is described in detail in the next section. For the moment it is only necessary to note that the projected search chooses an $\alpha_k > 0$ such that $q(x_{k+1}) < q(x_k)$, and that more than one constraint may be added to the active set whenever $\alpha_k > \mu_k$.

If the iterate x_{k+1} generated by the conjugate gradient method appears to be in the face which contains the solution, then this face is explored further. The decision to continue the conjugate gradient method is based on the observation that if x is on the face that contains the solution, then the *binding set*

$$\mathcal{B}(x) = \{i : x_i = l_i \text{ and } \partial_i q(x) \geq 0, \text{ or } x_i = u_i \text{ and } \partial_i q(x) \leq 0\}$$

agrees with the active set $\mathcal{A}(x)$. Thus, if the conjugate gradient method produces an iterate x_{k+1} such that $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$, then we continue exploring this face with the conjugate gradient method. The condition $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$ does not guarantee that x_{k+1} is in the face which contains the solution. If the current iterate is not in the face which contains the solution then the finite termination properties

of the conjugate gradient method guarantee that it eventually generates an iterate which violates the condition $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$.

We can continue exploring the current face by saving the last iterate w_j and the search direction s_j generated by the conjugate gradient method. Setting the starting point to w_j and the starting direction to s_j preserves previous information, and thus, the finite termination properties of the conjugate gradient method.

Once the conjugate gradient algorithm has explored a face, standard algorithms use multiplier estimates to choose a different face. However, there are theoretical and numerical reasons for using the gradient projection method.

For the quadratic programming problem (1.1), the gradient projection method generates a sequence $\{y_j\}$ by setting

$$y_{j+1} = P[y_j - \alpha_j \nabla q(y_j)]$$

where P is the projection (2.6) into Ω and $\alpha_j > 0$ is chosen by a projected search so that $q(y_{j+1}) < q(y_j)$. The gradient projection method is used to select a new face as follows: Given x_k, we generate iterates by the gradient projection method with $y_0 = x_k$. If for some fixed constant $\eta_2 > 0$ either of the two tests

$$\mathcal{A}(y_j) = \mathcal{A}(y_{j-1}), \tag{3.7}$$

$$q(y_{j-1}) - q(y_j) \le \eta_2 \max\{q(y_{l-1}) - q(y_l) : 1 \le l < j\} \tag{3.8}$$

is satisfied, then we explore the face of the feasible set that contains y_{j_k}, where j_k is the first index j that satisfies (3.7) or (3.8).

The justification of test (3.7) is based on the convergence properties of the gradient projection method as discussed by Bertsekas [1], Dunn [14], Calamai and Moré [5], and Burke and Moré [4]. These results show that for nondegenerate problems there is a neighborhood of the solution such that (3.7) holds whenever x_k belongs to this neighborhood. Since on nondegenerate problems the active set is only guaranteed to settle down in a neighborhood of the solution, there is a need for test (3.8) which detects when the gradient projection method is not making sufficient progress.

The use of the gradient projection method to choose a face of the feasible set has appeared in several optimization algorithms. For example, Moré and Toraldo [23] use (3.7), but replace (3.8) with a fixed limit on the number of consecutive gradient projection iterations. The use of (3.8) is natural and leads to a more efficient algorithm. Wright [26] eliminates (3.8) and replaces (3.7) with $\mathcal{B}(y_j) = \mathcal{B}(y_{j-1})$. As noted in the introduction, eliminating (3.8) completely can be inefficient.

We have described how the conjugate gradient method explores the face of the feasible region defined by the current iterate and how the gradient projection method chooses a different face. The following algorithm summarizes the discussion.

Algorithm GPCG.

For $k = 0, \cdots$ until convergence.

Generate gradient projection iterates $y_0, y_1, \cdots$ with $y_0 = x_k$. Set $x_k \leftarrow y_{j_k}$, where j_k is the first index j that satisfies (3.7) or (3.8).

Generate conjugate gradient iterates $w_0, w_1, \cdots$ with $w_0 = 0$. Set $d_k = Z_k w_{j_k}$, where j_k is the first index j that satisfies (3.4). Use a projected search to define x_{k+1} by (3.6). If $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$, continue the conjugate gradient method.

Each iteration of algorithm GPCG consists of choosing a face by the gradient projection algorithm and exploring that face by the conjugate gradient algorithm. We claim that each iteration of algorithm GPCG can be done in a finite number of steps. The gradient projection algorithm determines a suitable y_{j_k} in a finite number of steps because (3.8) is eventually satisfied provided q is bounded below on the feasible set. Also note that if the conjugate gradient method does not terminate, then it eventually generates an iterate such that $\mathcal{B}(x_{k+1}) \neq \mathcal{A}(x_{k+1})$.

We also claim that if algorithm GPCG terminates in a finite number of steps then it produces the answer to problem (1.1). This claim is a standard result for the gradient projection method. For the conjugate gradient method note that termination can only occur if the conjugate gradient method produces an iterate with $r_k = 0$ and $\mathcal{A}(x_k) = \mathcal{B}(x_k)$. These two conditions imply that x_k is the solution of problem (1.1).

4. Projected searches. The projected search in algorithm GPCG requires an $\alpha_k > 0$, which produces a sufficient decrease in the function $\phi_k : \mathbb{R} \to \mathbb{R}$ defined by

$$\phi_k(\alpha) = q\left(P\left[x_k + \alpha d_k\right]\right),$$

where P is the projection (2.6) into the bound constrained set (2.1), and d_k is the search direction. The *sufficient decrease* condition requires that $\alpha_k > 0$ satisfy

$$\phi_k(\alpha_k) \leq \phi_k(0) + \mu\langle \nabla q(x_k), P\left[x_k + \alpha_k d_k\right] - x_k \rangle \tag{4.1}$$

for some constant $\mu \in (0, \frac{1}{2})$. In this section we show that the sufficient decrease condition can be satisfied for the two choices of d_k in algorithm GPCG. Recall that for the gradient projection method $d_k = -\nabla q(x_k)$, while for the conjugate gradient method $d_k = Z_k w_{j_k}$ where w_{j_k} is produced by the conjugate gradient method for problem (3.2).

Note that ϕ_k is a continuous, piecewise quadratic function because $P\left[x_k + \alpha d_k\right]$ is a linear function of α on any interval on which the active set of $P\left[x_k + \alpha d_k\right]$ is unchanged. Since q is strictly convex, ϕ_k is piecewise strictly convex. Breakpoints for ϕ_k can be defined by computing

$$0 = \beta_0 < \beta_1 < \cdots < \beta_p < \beta_{p+1} = +\infty$$

so that $\mathcal{A}(P\left[x_k + \alpha d_k\right])$ is constant on the intervals (β_i, β_{i+1}) for $i = 0, \cdots, p$. We allow infinite values for either l_i or u_i, and thus we may have $p = 0$ in the definition of breakpoints.

When $d_k = Z_k w_{j_k}$, the choice of Z_k guarantees that $x_k + \alpha d_k$ is feasible for α in $[0, \beta_1]$, and thus (4.1) can be satisfied for all $\alpha_k > 0$ sufficiently small if $\phi_k'(0) < 0$. Note that for this choice of d_k, the first breakpoint β_1 coincides with μ_k as defined by (3.5). Hence, if r_k is the reduced gradient defined by (3.3), then

$$\phi_k'(0) = \nabla q(x_k)^T d_k = r_k^T w_{j_k}.$$

Since w_{j_k} is a descent direction in the conjugate gradient method, this shows that $\phi_k'(0) < 0$ unless $r_k = 0$.

The proof that (4.1) can be satisfied if $d_k = -\nabla q(x_k)$ is similar. The main observation needed in this case is that if r_k is the projected gradient (2.3) so that

$$r_k = \nabla_\Omega q(x_k),$$

then $P(x_k - \alpha \nabla q(x_k)) = x_k - \alpha r_k$ for α in $[0, \beta_1]$, and

$$\phi_k(\alpha) = q(x_k - \alpha r_k), \qquad \alpha \in [0, \beta_1]. \tag{4.2}$$

Thus (4.1) can be satisfied for all $\alpha_k > 0$ sufficiently small if $\nabla q(x_k)^T r_k > 0$. Since

$$\nabla q(x_k)^T r_k = \|r_k\|^2,$$

this shows that $\nabla q(x_k)^T r_k > 0$ unless x_k is the solution of problem (1.1). Note that for this choice of d_k the path $x_k + \alpha d_k$ may not be feasible for any $\alpha > 0$. Indeed, $\mathcal{A}(P[x_k + \alpha d_k])$ is the binding set $\mathcal{B}(x_k)$ for α in $(0, \beta_1)$.

The projected search algorithm generates a decreasing sequence $\{\alpha_k^{(l)}\}$ of positive trial values such that

$$\alpha_k^{(l+1)} \in [\gamma_1 \alpha_k^{(l)}, \gamma_2 \alpha_k^{(l)}], \qquad 0 < \gamma_1 < \gamma_2 < 1.$$

We choose an $\alpha_k^{(0)}$ bounded away from zero and set α_k to the first trial value that satisfies the sufficient decrease condition. One of the advantages of this procedure is that it produces an acceptable α_k with a finite number of evaluations of q.

The initial trial value of α_k is the minimizer of the quadratic function that represents ϕ_k in $[0, \beta_1]$. Thus the initial trial value of α_k is

$$\alpha_k^{(0)} = -\frac{\phi_k'(0)}{\phi_k''(0)}.$$

An expression for the initial trial value of α_k is obtained by noting that $x_k + \alpha d_k$ is feasible for α in $[0, \beta_1]$ when $d_k = Z_k w_k$, and thus the conjugacy properties of the conjugate gradient algorithm imply that

$$\alpha_k^{(0)} = -\frac{r_k^T w_{j_k}}{w_{j_k}^T A_k w_{j_k}} = 1 \tag{4.3}$$

for this choice of d_k. For $d_k = -\nabla q(x_k)$ it is only necessary to note that (4.2) implies that

$$\alpha_k^{(0)} = \frac{\|r_k\|^2}{r_k^T A_k r_k}. \tag{4.4}$$

For either choice of d_k, if $\alpha_k^{(0)} < \beta_1$ then the initial trial value satisfies the sufficient decrease condition (2.4) with $\mu \leq \frac{1}{2}$, and thus becomes the chosen α_k.

Given an initial trial value we compute the minimizer α_k^* of the quadratic that interpolates $\phi_k(0)$, $\phi_k'(0)$, $\phi_k(\alpha_k)$, and we use the maximum of β_1 and

$$\operatorname{mid}(\tfrac{1}{100}\alpha_k, \alpha_k^*, \tfrac{1}{2}\alpha_k) \tag{4.5}$$

as the new estimate, where $\operatorname{mid}(\cdot, \cdot, \cdot)$ has been defined in connection with (2.6). This procedure is repeated until the sufficient decrease condition holds. Termination in a finite number of steps is guaranteed because β_1 satisfies the sufficient decrease condition (4.1) with $\mu \leq \frac{1}{2}$ whenever $\alpha_k^{(0)} < \beta_1$.

The projected search described above is based on the work of Dembo and Tulowitzki [13] and Moré and Toraldo [23]. The use of (4.5) to update the trial values of α_k is a standard quadratic interpolation scheme with safeguards to avoid large

corrections to α_k. Other authors use simpler schemes (for example, always use $\frac{1}{2}\alpha_k$ as the new trial value), but (4.5) is likely to be better.

Many optimization algorithms use projected searches, but the precise choice of α_k usually depends on the algorithm. For example, Conn, Gould, and Toint [10] use a projected search but choose α_k as the first local minimizer of ϕ_k. A disadvantage of this choice is that it may require the evaluation of the function at every breakpoint. Since function evaluations tend to be expensive, this is not desirable. From a theoretical viewpoint, this choice has only been shown to be satisfactory for bound constrained problems. In contrast, there are convergence results for a wide class of algorithms with the projected search described in this section. See, for example, Bertsekas [1], Dunn [14], Calamai and Moré [5], and Burke and Moré [4].

5. Convergence results. The main theoretical results for algorithm GPCG are that GPCG converges for strictly convex problems, and that if the problem is nondegenerate, GPCG terminates in a finite number of iterations.

THEOREM 5.1. *Let $q : \mathbb{R}^n \to \mathbb{R}$ be a strictly convex quadratic. If $\{x_k\}$ is the sequence generated by algorithm* GPCG *for problem* (1.1), *then either $\{x_k\}$ terminates at the solution x^* of problem* (1.1) *in a finite number of steps, or $\{x_k\}$ converges to x^*.*

Proof. We have already noted that if GPCG terminates in a finite number of steps, then it must terminate at a stationary point of problem (2.1). Since q is strictly convex the only stationary point of problem (2.1) is the solution x^*.

We now consider the case in which algorithm GPCG generates an infinite sequence $\{x_k\}$. Since q is a strictly convex quadratic and $\{q(x_k)\}$ is a nonincreasing sequence, $\{x_k\}$ is bounded. In particular, if $\mathcal{K}_{GP}$ is the set of iterates generated by the gradient projection method, $\mathcal{K}_{GP}$ is infinite and $\{x_k : k \in \mathcal{K}_{GP}\}$ is bounded. Theorem 5.2 of Calamai and Moré [5] guarantees that every limit point of $\{x_k : k \in \mathcal{K}_{GP}\}$ is a stationary point of problem (1.1). This result implies that there is a subsequence of $\{x_k\}$ that converges to x^*, and thus $\{q(x_k)\}$ converges to $q(x^*)$. Now note that inequality (2.2) holds because x^* is a stationary point, and thus

$$q(x_k) - q(x^*) = \langle \nabla q(x^*), x_k - x^* \rangle + \tfrac{1}{2}\langle \nabla^2 q(x^*)(x_k - x^*), x_k - x^* \rangle \geq \frac{\sigma}{2}\|x_k - x^*\|^2,$$

where σ is a lower bound for the eigenvalues of the Hessian. Since $\{q(x_k)\}$ converges to $q(x^*)$, this inequality shows that $\{x_k\}$ converges to x^*, as desired. □

Theorem 5.1 does not make strong use of the properties of the conjugate gradient method. Convergence of GPCG depends on the convergence properties of the gradient projection method, and on the result that the conjugate gradient method produces iterates such that $q(x_{k+1}) \leq q(x_k)$.

We now show that finite termination of GPCG depends on the finite termination properties of the conjugate gradient method on subproblems, and on the identification properties of the gradient projection method. These identification properties have been explored by Bertsekas [1], [2], Dunn [14], [15], Calamai and Moré [5], and Burke and Moré [4]. In the following theorem we use the results as formulated by Calamai and Moré.

THEOREM 5.2. *Let $q : \mathbb{R}^n \to \mathbb{R}$ be a strictly convex quadratic. If the solution x^* of problem* (1.1) *is nondegenerate in the sense that*

$$\partial q_i(x^*) \neq 0, \qquad i \in \mathcal{A}(x^*),$$

then algorithm GPCG *terminates at the solution x^* in a finite number of steps.*

Proof. We claim that if the conjugate gradient method in algorithm GPCG is given an x_k with $\mathcal{A}(x_k) = \mathcal{A}(x^*)$ and sufficiently close to x^*, then the conjugate gradient method produces the solution x^* in a finite number of steps. We establish this claim by first noting that since the conjugate gradient method does not violate any active constraints,

$$\mathcal{A}(x^*) = \mathcal{A}(x_k) \subset \mathcal{A}(x_{k+1}).$$

Since $\mathcal{A}(x) \subset \mathcal{A}(x^*)$ for any feasible x sufficiently close to x^*, we have shown that $\mathcal{A}(x_{k+1}) = \mathcal{A}(x^*)$. Now consider any feasible x sufficiently close to x^*. The continuity of ∇q and the nondegeneracy assumption imply that $\partial q_i(x) \neq 0$ for $i \in \mathcal{A}(x^*)$. Thus, $\mathcal{A}(x) = \mathcal{A}(x^*)$ implies that $\mathcal{B}(x) = \mathcal{A}(x^*) = \mathcal{A}(x)$. In particular, since we have shown that $\mathcal{A}(x_{k+1}) = \mathcal{A}(x^*)$, we obtain that $\mathcal{B}(x_{k+1}) = \mathcal{A}(x_{k+1})$. This implies that the conjugate gradient method continues to explore the current face, and thus produces the solution x^* in a finite number of steps.

We complete the proof by appealing to Theorems 4.1 and 5.2 of Calamai and Moré [5]. These results show that if $\mathcal{K}_{GP}$ is the set of iterates generated by the gradient projection method, then $\mathcal{A}(y_{j_k}) = \mathcal{A}(x^*)$ whenever x_k is sufficiently close to x^* and $k \in \mathcal{K}_{GP}$. Thus the conjugate gradient method in algorithm GPCG is eventually given an x_k with $\mathcal{A}(x_k) = \mathcal{A}(x^*)$ and sufficiently close to x^*. □

6. Test problems. We consider three model problems: the obstacle problem, the elastic-plastic torsion problem, and the journal bearing problem. These problems can be formulated as minimization problems of the form

$$\min\{q(v) : v \in K\}, \tag{6.1}$$

where $q : K \to \mathbb{R}$ is a quadratic over a closed convex set K in a Hilbert space. In this section we describe the continuous version of these minimization problems and the finite element approximations which lead to finite dimensional problems.

The continuous version of these problems is described in terms of a bounded open set $\mathcal{D}$ in $\mathbb{R}^d$ with a reasonably smooth boundary $\partial\mathcal{D}$, and the space $H_0^1(\mathcal{D})$ of all functions with compact support in $\mathcal{D}$ such that v and $\|\nabla v\|^2$ belong to $L^2(\mathcal{D})$. Note that $v = 0$ on $\partial\mathcal{D}$ for any $v \in H_0^1(\mathcal{D})$.

In the *obstacle* problem we are given obstacles v_l and v_u such that $v_l \leq 0$ and $v_u \geq 0$ on $\partial\mathcal{D}$, and a function $f \in L^2(\mathcal{D})$. The obstacle problem is of the form (6.1) with

$$q(v) = \tfrac{1}{2}\int_{\mathcal{D}} \|\nabla v\|^2 d\mathcal{D} - \int_{\mathcal{D}} f v d\mathcal{D} \tag{6.2}$$

and

$$K = \{v \in H_0^1(\mathcal{D}) : v_l \leq v \leq v_u \text{ on } \mathcal{D}\}.$$

Although the force function f can be nonlinear, in our test problems we assume that $f \equiv c$ for some constant c. The obstacle problem and its physical interpretation are discussed, for example, in Ciarlet [7, pp. 287–296]. An interesting aspect of the obstacle problem is that the solution is not usually twice continuously differentiable even if v_l, v_u, and f are infinitely differentiable. This aspect of the obstacle problem is shared by the other two problems in this section.

The elastic-plastic *torsion* problem can be formulated as a minimization problem of the form (6.1) where q is defined by (6.2) with $f \equiv c$ for some constant c, and

$$K = \{v \in H_0^1(\mathcal{D}) : |v(x)| \leq \operatorname{dist}(x, \partial\mathcal{D}),\ x \in \mathcal{D}\}$$

with $\operatorname{dist}(\cdot, \partial\mathcal{D})$ the distance function to the boundary of $\mathcal{D}$. This formulation and the physical interpretation of the torsion problem are discussed, for example, in Glowinski [16, pp. 158–162].

The obstacle problem and the torsion problems have been formulated for any reasonable open set in $\mathbb{R}^d$, although applications are restricted to the case $d \leq 3$. In the *journal bearing* problem we are concerned with a particular set in $\mathbb{R}^2$. For this problem $\mathcal{D}$ is the rectangle

$$\mathcal{D} = \{(\theta, y) : 0 < \theta < 2\pi,\ 0 < y < 2b\}. \tag{6.3}$$

The journal bearing problem is of the form (6.1), where

$$q(v) = \tfrac{1}{2}\int_{\mathcal{D}} (1 + \epsilon\cos\theta)^3 \|\nabla v\|^2 d\mathcal{D} - \epsilon\int_{\mathcal{D}} \sin\theta v d\mathcal{D}, \tag{6.4}$$

and

$$K = \{v \in H_0^1(\mathcal{D}) : v \geq 0 \text{ on } \mathcal{D}\}.$$

The parameter $\epsilon \in (0, 1)$ is the eccentricity of the journal bearing. In the interest of simplicity, we have followed Lin and Cryer [19] in neglecting the periodicity conditions in the journal bearing problem as formulated by Cimatti [8]. The physical interpretation of the journal bearing problem is discussed by Capriz and Cimatti [6].

We now describe finite element approximations to the above problems. These approximations are standard, but we need to be explicit in order to define the test problem.

The general approach is to triangulate $\mathcal{D}$ and to replace the minimization of q over $H_0^1(\mathcal{D})$ by the minimization of q over the set of piecewise linear functions that satisfy the constraints specified by K. The finite element approximations thus give rise to a finite dimensional minimization problem whose variables are the values of the piecewise linear function at the vertices of the triangulation.

Let $\mathcal{D}$ be a rectangle in $\mathbb{R}^2$ and set $\mathcal{D} = (d_1, d_2) \times (d_3, d_4)$. Vertices $z_{i,j} \in \mathbb{R}^2$ for a triangulation of $\mathcal{D}$ are obtained by choosing grid spacings h_x and h_y and defining grid points

$$z_{i,j} = (d_1 + ih_x, d_3 + jh_y), \qquad 0 \leq i \leq n_x + 1, \quad 0 \leq j \leq n_y + 1 \tag{6.5}$$

such that $z_{n_x+1,n_y+1} = (d_2, d_4)$. The triangulation consists of triangular elements T_U with vertices at

$$z_{i,j}, \qquad z_{i+1,j}, \qquad z_{i,j+1},$$

and triangular elements T_L with vertices at

$$z_{i,j}, \qquad z_{i-1,j}, \qquad z_{i,j-1}.$$

Consider the piecewise linear function v with values $v_{i,j}$ at $z_{i,j}$. Since ∇v is constant over any element, a computation shows that

$$\int_{T_U} \|\nabla v\|^2 d\mathcal{D} = \frac{h_x h_y}{2}\left\{\left(\frac{v_{i+1,j} - v_{i,j}}{h_x}\right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y}\right)^2\right\},$$

and that

$$\int_{T_L} \|\nabla v\|^2 d\mathcal{D} = \frac{h_x h_y}{2} \left\{ \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\},$$

where $v_{i,j} = 0$ if $z_{i,j} \in \partial\mathcal{D}$. Similarly, since v is linear over any element,

$$\int_{T_U} v d\mathcal{D} = \frac{h_x h_y}{6} \{v_{i+1,j} + v_{i,j+1} + v_{i,j}\}, \quad \int_{T_L} v d\mathcal{D} = \frac{h_x h_y}{6} \{v_{i-1,j} + v_{i,j-1} + v_{i,j}\}.$$

Thus, assembling all the contributions yields

$$\int_{\mathcal{D}} v d\mathcal{D} = h_x h_y \sum v_{i,j}.$$

The above computations show that if q is defined by (6.2) with $f \equiv c$ for some constant c and v is the piecewise linear function with values $v_{i,j}$ at $z_{i,j}$ then

$$q(v) = \frac{1}{4} \sum q_{i,j}(v) - c h_x h_y \sum v_{i,j}, \tag{6.6}$$

where

$$q_{i,j}(v) = h_x h_y \left\{ \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right.$$

$$\left. + \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}.$$

When $h_x = h_y = h$ we can express the quadratic q in matrix notation by noting that if we identify the piecewise linear function v with the vector in $\mathbb{R}^{n_x n_y}$ with components $v_{i,j}$ in the standard row-wise ordering, then

$$q(v) = \tfrac{1}{2} v^T A v - b^T v,$$

where b is the vector with $b_{i,j} = ch^2$, and A is the usual block tridiagonal matrix (with diagonal entries of 4 and off-diagonal entries of -1) arising from a difference approximation to the Laplacian operator.

We have defined finite dimensional approximations to the obstacle problem with $f \equiv c$ and the torsion problem. In both cases the finite element approximation leads to a minimization problem of the form

$$\min\{q(v) : v \in \Omega\}, \tag{6.7}$$

where q is the quadratic defined by (6.6) and Ω is a convex set defined by bounds. For the obstacle problem

$$\Omega = \{v \in \mathbb{R}^{n_x n_y} : l_{i,j} \le v_{i,j} \le u_{i,j}\}, \tag{6.8}$$

where $l_{i,j}$ and $u_{i,j}$ are, respectively, the values of v_l and v_u at $z_{i,j}$. For the torsion problem

$$\Omega = \{v \in \mathbb{R}^{n_x n_y} : |v_{i,j}| \le d_{i,j}\} \tag{6.9}$$

where $d_{i,j}$ is the value of $\operatorname{dist}(\cdot, \partial\mathcal{D})$ at $z_{i,j}$.

A finite element approximation to the journal bearing problem is obtained by an extension of the above considerations to deal with integrals of the form

$$\int_{\mathcal{D}} w_q \|\nabla v\|^2 d\mathcal{D}, \qquad \int_{\mathcal{D}} w_l v d\mathcal{D},$$

where $w_q : \mathcal{D} \to \mathbb{R}$ and $w_l : \mathcal{D} \to \mathbb{R}$ are functions defined on the rectangle $\mathcal{D}$. If v is the piecewise linear function with values $v_{i,j}$ at $z_{i,j}$ then

$$\int_{T_U} w_q \|\nabla v\|^2 d\mathcal{D} = \mu_{i,j} \left\{ \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\}$$

where the constant $\mu_{i,j}$ is the integral of the function w_q over T_U. In the journal bearing problem, $w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$, and thus it is possible to obtain an exact expression for $\mu_{i,j}$. In our test problem, however, we use the trapezoidal rule to approximate the integral, and this leads to

$$\mu_{i,j} := \frac{h_x h_y}{6} \left\{ w_q(z_{i,j}) + w_q(z_{i+1,j}) + w_q(z_{i,j+1}) \right\}. \tag{6.10}$$

Similarly,

$$\int_{T_L} w_q \|\nabla v\|^2 d\mathcal{D} = \lambda_{i,j} \left\{ \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}$$

where

$$\lambda_{i,j} := \frac{h_x h_y}{6} \left\{ w_q(z_{i,j}) + w_q(z_{i-1,j}) + w_q(z_{i,j-1}) \right\}. \tag{6.11}$$

The integral of $w_l v$ over any element can be evaluated exactly for the journal bearing problem because $w_l(\xi_1, \xi_2) = \sin \xi_1$ and v is linear on any element. In our test problem we approximate this integral with the trapezoidal rule, and thus

$$\int_{\mathcal{D}} w_l v d\mathcal{D} := h_x h_y \sum w_l(z_{i,j}) v_{i,j}.$$

The above approximations can be used, in particular, on the journal bearing functional defined by (6.4). We obtain a minimization problem of the form (6.7) with Ω the set of v with $v \geq 0$ and with

$$q(v) = \frac{1}{2} \sum q_{i,j}(v) - \epsilon h_x h_y \sum w_l(z_{i,j}) v_{i,j}, \tag{6.12}$$

where $w_l(\xi_1, \xi_2) = \sin \xi_1$, and the quadratic $q_{i,j}$ is defined by

$$q_{i,j}(v) = \mu_{i,j} \left\{ \left(\frac{v_{i+1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j+1} - v_{i,j}}{h_y} \right)^2 \right\}$$

$$+ \lambda_{i,j} \left\{ \left(\frac{v_{i-1,j} - v_{i,j}}{h_x} \right)^2 + \left(\frac{v_{i,j-1} - v_{i,j}}{h_y} \right)^2 \right\}.$$

Recall that the constants $\mu_{i,j}$ and $\lambda_{i,j}$ are defined by (6.10) and (6.11), and that $w_q(\xi_1, \xi_2) = (1 + \epsilon \cos \xi_1)^3$.

7. Numerical results. We consider the three problems described in §6 and examine the performance of algorithm GPCG on these problems. Our main concern is with the performance of the algorithm on problems with a large number of variables.

We study the behavior of the GPCG algorithm in terms of the number of iterations required to satisfy the convergence criterion. This number is of interest because it represents the number of faces that are searched before the face which contains the solution is found. Our results show that even for large problems, the number of iterations is small; in all cases fewer than 15.

Although the number of iterations is of interest, the total amount of work is a crucial ingredient in the performance evaluation of GPCG. A reasonable measure of the amount of work required by GPCG is the number of function-gradient evaluations and Hessian-vector products needed to solve the problem. In some papers only the number of iterations is mentioned, and this does not provide a reasonable measure of the amount of work.

In the numerical results n_f is the number of function-gradient evaluations and n_h is the number of Hessian-vector products. We provide the averages of n_f and n_h per iteration. We have attempted to simplify the performance evaluation of GPCG by just providing the total number of function-gradient evaluations and Hessian-vector products. Note, however, that the cost of a Hessian-vector product depends on the number of free variables because the number of nonzero components in the vector involved in the Hessian-vector product is at most the number of free variables during the current iteration. Thus, the faster algorithm is not necessarily the algorithm with the least number of function-gradient evaluations and Hessian-vector products. This point should be kept in mind during the evaluation of the numerical results of algorithm GPCG.

In addition to the averages of n_f and n_h, there are other statistics which could be of interest. In some cases, bounds for these statistics can be obtained from n_f and n_h. Consider, for example, the number of gradient projection iterations and the number of conjugate gradient iterations. Since an iteration of either the gradient projection method or the conjugate gradient method requires a Hessian-vector product,

$$n_{CG} + n_{GP} = n_h,$$

where n_{CG} is the number of conjugate gradient iterations, and n_{GP} is the number of gradient projection iterations. Moreover, since the projected search of the conjugate gradient method requires at least one function-gradient evaluation,

$$n_{GP} \leq n_f - n_{iter},$$

where n_{iter} is the number of iterations. In our numerical results we will find that n_f is smaller than n_h (usually by at least a factor of 3), so the above relationships give us an indication of the relative values of n_{CG} and n_{GP}. For example, if on a problem we obtain $n_{iter} = 10$, $n_f = 45$, and $n_h = 157$, then $n_{GP} \leq 35$ and $n_{CG} \geq 122$.

Most of the ingredients of algorithm GPCG are described in §§3 and 4. The results in this section were obtained with the following parameter settings:

$$\eta_1 = 0.1, \quad \eta_2 = 0.25, \quad \mu = 0.1, \quad \tau = 10^{-5}.$$

Recall that η_1 appears in the sufficient progress test (3.4) of the conjugate gradient method, η_2 in the sufficient progress test (3.8) of the gradient projection method, μ in the sufficient decrease condition (4.1) of the projected search, and τ in the convergence criterion (2.4) of algorithm GPCG.

The test problems have been described in §6. Since we are interested in large-scale problems, we have chosen problems with 5000 to 15,000 variables. This was done by choosing $n_x = n_y$ in the triangulation (6.5) from the set $\{75, 100, 125\}$. This leads to problems with dimension $n = n_x n_y$.

The numerical results were done in double precision (16 decimal places) on the Alliant FX/8 at the Advanced Computing Research Facility of Argonne National Laboratory.

7.1. The obstacle problem. The discrete obstacle problem is of the form (6.7) where q is the quadratic (6.6) and Ω is the set (6.8) with $l_{i,j}$ and $u_{i,j}$ being, respectively, the values of v_l and v_u at the grid points of the triangulation of $\mathcal{D}$. In our problem $\mathcal{D} = (0,1) \times (0,1)$. The only remaining ingredients are the specification of the obstacles v_l and v_u, the constant c, and the starting point x_0.

We have chosen two of the obstacle problems from Dembo and Tulowitzki [13]. For obstacle problem A the obstacles v_l and v_u are

$$v_l(\xi_1, \xi_2) = \sin(3.2\xi_1)\sin(3.3\xi_2), \qquad v_u(\xi_1, \xi_2) = 2000,$$

while for obstacle problem B, they are

$$v_l(\xi_1, \xi_2) = (\sin(9.2\xi_1)\sin(9.3\xi_2))^3, \qquad v_u(\xi_1, \xi_2) = (\sin(9.2\xi_1)\sin(9.3\xi_2))^2 + 0.02.$$

Obstacle problem B has nontrivial lower and upper bounds in the sense that both lower and upper bounds are active at the solution. Obstacle problem A only has nontrivial lower bounds. Results obtained with these two problems are typical for the other obstacle problems in Dembo and Tulowitzki [13]. In both problems $c = 1$ is the force constant.

The starting point of problem A is either the vector l specified by v_l or the vector e with all components set to unity. These choices test the algorithm with a starting point with all constraints active, and with a starting point in the interior of the feasible region. For similar reasons, the starting points for obstacle problem B are the lower bound vector l, the upper bound vector u, and the vector $\frac{1}{2}(l + u)$.

The results for the obstacle problem appear in Tables 7.1 and 7.2. These results show that algorithm GPCG satisfies the convergence criteria in a few iterations (fewer than 15), and with a reasonable number of function-gradient evaluations and Hessian-vector products per iteration. As noted above, the number of iterations is of interest because it represents the number of faces that are searched before the face which contains the solution is found. We also note that the number of iterations would not have changed if we had used a tighter convergence criterion; the only difference would have been an increase in the averages for n_f and n_h.

If we only consider the number of iterations or the number of function-gradient evaluations and Hessian-vector products, the performance of algorithm GPCG with respect to the starting point does not seem to be predictable; for problem A the algorithm prefers a starting point with all constraints active, while for problem B better behavior is obtained when x_0 is on the interior. However, if we consider the computing time, then our implementation of GPCG always requires the least amount of time when the starting point is a vertex of the feasible region. We can explain this observation by noting that when the starting point is on the interior, it is quite likely that the conjugate gradient method will be used to explore the interior of the feasible set. Since the cost of a Hessian-vector product depends on the number of free variables, the cost of exploring the interior of the feasible set is high, and tends

TABLE 7.1
Obstacle problem A.

n	nfree	x_0	Iterations	n_f/iter	n_h/iter
5625	2122	l	9	4.1	13.2
5625	2122	e	11	5.6	13.3
10000	3843	l	10	4.5	15.7
10000	3843	e	12	4.1	17.6
15626	6108	l	11	4.3	17.1
15626	6108	e	14	3.8	17.0

TABLE 7.2
Obstacle problem B.

n	nfree	x_0	Iterations	n_f/iter	n_h/iter
5625	4171	l	9	4.2	14.5
5625	4171	u	8	4.2	15.0
5625	4171	$\frac{1}{2}(l+u)$	5	4.8	18.0
10000	7588	l	8	4.3	21.7
10000	7588	u	10	4.5	16.8
10000	7588	$\frac{1}{2}(l+u)$	7	4.2	18.2
15626	11990	l	12	3.9	19.5
15626	11990	u	11	4.4	19.6
15625	11990	$\frac{1}{2}(l+u)$	8	4.2	22.1

to offset any gains made in reducing the number of function-gradient evaluations and Hessian-vector products.

7.2. The elastic-plastic torsion problem. The elastic-plastic torsion problem is of the form (6.7) where q is the quadratic (6.6) and Ω is the set (6.9). In our specification of the torsion problem we follow many of the suggestions of O'Leary [24]. In particular, $\mathcal{D} = (0,1) \times (0,1)$.

The starting point x_0 is either the origin, or the vector u of upper bounds. The vector l of lower bounds is not a reasonable starting point because it can be shown that if x_c^* is the solution of the torsion problem for a given c, then

$$\lim_{c \to \infty} x_c^* = u.$$

The convergence of x_c^* to u can be seen clearly by comparing the plot of the solution x_c^* for $c = 5$ in Fig. 7.1 with that for $c = 20$ in Fig. 7.2.

Numerical results for the torsion problem appear in Tables 7.3–7.5. The results for $c = 5$ are similar to those obtained for the obstacle problem A. In particular, note that algorithm GPCG satisfies the convergence criteria for the torsion problem with less than 15 iterations, and with a small number of function-gradient evaluations and Hessian-vector products per iteration.

The numerical results for the torsion problem improve with increasing c. This is to be expected because as c increases, the linear term in the function tends to dominate. This also explains the decrease in the percentage of free variables at the solution as c increases; the solution of a linear problem is always at a vertex.

In terms of iterations or number of function-gradient evaluations and Hessian-vector products, the behavior of algorithm GPCG with respect to the starting point

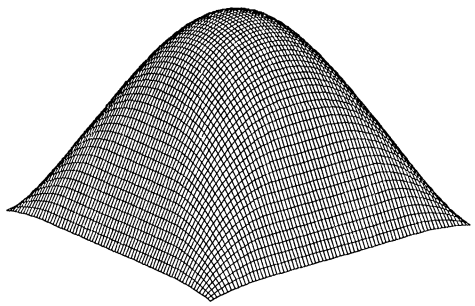


FIG. 7.1. *Torsion problem with* $c = 5$.

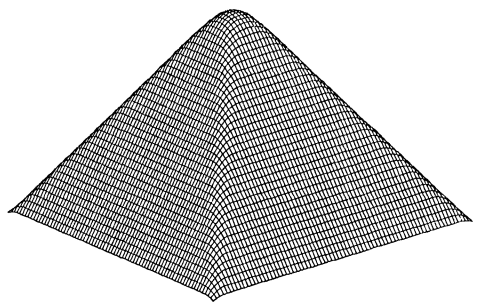


FIG. 7.2. *Torsion problem with* $c = 20$.

changes with c. For $c = 5$ the algorithm prefers $x_0 = 0$, while for larger values of c the starting point $x_0 = u$ gives better results. Just as in the obstacle problems, the computing time for our implementation of GPCG is always in favor of $x_0 = u$ as a starting point.

Note that for $c = 5, 10$ and any starting point, the number of iterations increase slightly as the dimension n increases, while for $c = 20$, the number of iterations does not change as the dimension increases. This is an important aspect of the numerical results because it suggests that larger problems can be solved with little additional computational effort.

7.3. The journal bearing problem. The journal bearing problem is of the form (6.7) where q is the quadratic (6.12) and Ω is the set of v with $v \geq 0$. From a numerical point of view, this problem is of interest because good numerical results

can only be obtained if the conjugate gradient method is used with a preconditioner. The numerical results for the journal bearing problem were obtained by considering the equivalent problem

$$\min\{\hat{q}(v) : v \geq 0\}, \qquad \hat{q}(v) = q(D^{-1}v),$$

where D is the diagonal matrix chosen so that the the Hessian matrix of $\hat{q}$ has unit entries on the diagonal, and q is the quadratic (6.12).

TABLE 7.3
Elastic-plastic torsion problem with $c = 5$.

n	nfree	x_0	Iterations	n_f/iter	n_h/iter
5625	3953	0	9	4.4	27.1
5625	3953	u	11	4.5	20.1
10000	7016	0	11	4.9	27.9
10000	7016	u	14	4.5	23.3
15625	10961	0	12	5.0	33.5
15625	10961	u	14	5.0	27.5

TABLE 7.4
Elastic-plastic torsion problem with $c = 10$.

n	nfree	x_0	Iterations	n_f/iter	n_h/iter
5625	2073	0	8	4.2	17.5
5625	2073	u	7	4.4	15.8
10000	3632	0	8	4.5	21.8
10000	3632	u	7	4.7	19.8
15625	5789	0	10	4.5	21.7
15625	5789	u	9	4.4	18.5

TABLE 7.5
Elastic-plastic torsion problem with $c = 20$.

n	nfree	x_0	Iterations	n_f/iter	n_h/iter
5625	1025	0	6	3.8	11.1
5625	1025	u	5	4.0	10.0
10000	1768	0	6	4.0	13.6
10000	1768	u	5	4.2	11.8
15625	9248	0	6	4.5	18.8
15625	9248	u	5	4.8	14.6

We varied the eccentricity ϵ, and used $\epsilon = 0.1, 0.5$ because they are typical values for the eccentricity. A plot of the solution for eccentricity $\epsilon = 0.1$ appears in Fig. 8.1. The main difference between the solution for $\epsilon = 0.1$ and $\epsilon = 0.5$ is that the number of active variables increases; otherwise the two solutions are quite similar.

We also varied the dimensions of the journal bearing by setting $b = 1, 10, 100$ in (6.3). In Tables 8.1 and 8.2 we only present the results for $b = 10$ because they are representative of results obtained for other values of b.

In the numerical results we used $x_0 = 0$ as the starting point. The results obtained with this starting point are similar to those obtained in the obstacle and torsion problems in the sense that algorithm GPCG satisfies the convergence criteria in a few iterations and with a small number of function-gradient evaluations and Hessian-vector products per iteration.

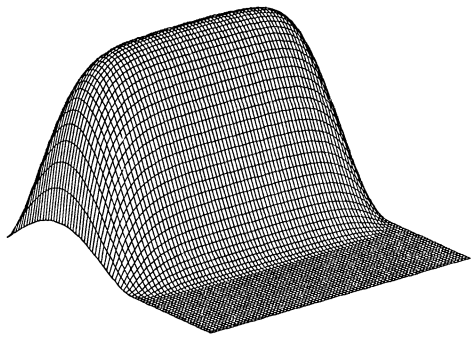


FIG. 8.1. *Journal bearing problem with* $\epsilon = 0.1$.

Note that as the eccentricity ϵ increases the number of free variables at the solution and the number of iterations required for convergence decreases. Thus the eccentricity ϵ plays a similar role to that of c in the torsion problem.

8. Final remarks. The numerical results for algorithm GPCG are quite interesting and merit further investigation. It would be interesting, for example, to extend these ideas to general nonlinear minimization problems subject to bound constraints.

A key issue that we have not addressed is the numerical comparison of our results with other algorithms in the literature. Numerical comparisons are difficult because most papers for problem (1.1) use different test problems. Moreover, even if the same problems are used, the results are stated in terms of iterations and not function-gradient evaluations.

We have only been able to make a rough comparison of our results with those of Dembo and Tulowitzki [13]. This required changing the termination criterion from (2.4) to

$$\|\nabla_\Omega q(x)\| \leq \tau,$$

and setting $\tau = 10^{-5}$. In their numerical results, Dembo and Tulowitzki considered four algorithms and concluded that algorithm CGP was the best overall method. On obstacle problem A with $n = 10,000$ algorithm CGP required 310 iterations when $x_0 = l$ and 681 iterations for $x_0 = e$. Each iteration of CGP requires a projected search, and each projected search requires one Hessian-vector product and at least one function-gradient evaluation. In contrast, algorithm GPCG requires 45 function-gradient evaluations and 157 Hessian-vector products for $x_0 = l$, and 56 function-gradient evaluations and 246 Hessian-vector products for $x_0 = e$.

Results obtained on the other problems from Dembo and Tulowitzki show similar improvements. Note that the small number of function-gradient evaluations required by GPCG is of significance because each evaluation of the function depends on the number of variables, while a Hessian-vector product depends on the number of free variables in the current iteration.

We hope that GPCG will be compared with other algorithms in the literature, and that the detailed description of the problems in §6 will encourage other researchers in this area to use these problems as tests for their algorithms.

TABLE 8.1
Journal bearing problem with $\epsilon = 0.1$.

n	nfree	Iterations	n_f/iter	n_h/iter
5625	3808	9	4.3	28.2
10000	6768	10	4.6	33.3
15625	10564	10	5.2	41.5

TABLE 8.2
Journal bearing problem with $\epsilon = 0.5$.

n	nfree	Iterations	n_f/iter	n_h/iter
5625	3359	5	5.0	31.6
10000	6040	7	4.7	31.0
15625	9426	7	5.1	40.2

Acknowledgment. This work benefited from discussions with Andreas Griewank, Jim Northrup, and Steve Wright.

REFERENCES

[1] D. P. BERTSEKAS, *On the Goldstein–Levitin–Polyak gradient projection method*, IEEE Trans. Automat. Control, 21 (1976), pp. 174–184.

[2] ———, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221–246.

[3] A. BJÖRCK, *A direct method for sparse least squares problems with lower and upper bounds*, Numer. Math., 54 (1988), pp. 19–32.

[4] J. V. BURKE AND J. J. MORÉ, *On the identification of active constraints*, SIAM J. Numer. Anal., 25 (1988), pp. 1197–1211.

[5] P. H. CALAMAI AND J. J. MORÉ, *Projected gradient methods for linearly constrained problems*, Math. Programming, 39 (1987), pp. 93–116.

[6] G. CAPRIZ AND G. CIMATTI, *Free boundary problems in the theory of hydrodynamic lubrication : A survey*, in Free Boundary Problems: Theory and Applications, A. Fasano and M. Primicerio, eds., Pitman, Boston, 1983.

[7] P. G. CIARLET, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[8] G. CIMATTI, *On a problem of the theory of lubrication governed by a variational inequality*, Appl. Math. Optim., 3 (1977), pp. 227–242.

[9] T. F. COLEMAN AND L. A. HULBERT, *A direct active set algorithm for large sparse quadratic programs with simple bounds*, Math. Programming, 45 (1989), pp. 373–406.

[10] A. R. CONN, N. I. M. GOULD, AND P. L. TOINT, *Testing a class of methods for solving minimization problems with simple bounds on the variables*, Math. Comp., 50 (1988), pp. 399–430.

[11] R. W. COTTLE AND M. S. GOHEEN, *A special class of large quadratic programs*, in Nonlinear Programming 3, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, 1978.

[12] R. W. COTTLE, G. H. GOLUB, AND R. S. SACHER, *On the solution of large, structured linear complementarity problems: The block partitioned case*, Appl. Math. Optim., 4 (1978), pp. 347–363.

[13] R. S. DEMBO AND U. TULOWITZKI, *On the minimization of quadratic functions subject to box constraints*, Working Paper 71, School of Organization and Management, Yale University, New Haven, CT, 1983.

[14] J. C. DUNN, *Global and asymptotic convergence rate estimates for a class of projected gradient processes*, SIAM J. Control Optim., 19 (1981), pp. 368–400.

[15] ———, *On the convergence of projected gradient processes to singular critical points*, J. Optim. Theory Appl., 55 (1987), pp. 203–216.

[16] R. GLOWINSKI, *Numerical Methods for Nonlinear Variational Problems*, Springer-Verlag, Berlin, New York, 1984.

[17] J. J. JÚDICE AND F. M. PIRES, *Bard-type methods for the linear complementarity problem with symmetric positive definite matrices*, IMA J. Math. Appl. Business and Industry, 2 (1989), pp. 51–68.

[18] ———, *Direct methods for convex quadratic programs subject to box constraints*, preprint, Universidada de Coimbra, Coimbra, Portugal, 1989.

[19] Y. LIN AND C. W. CRYER, *An alternating direction implicit algorithm for the solution of linear complementarity problems arising from free boundary problems*, Appl. Math. Optim., 13 (1985), pp. 1–17.

[20] Y. Y. LIN AND J. S. PANG, *Iterative methods for large convex quadratic programs: A survey*, SIAM J. Control Optim., 25 (1987), pp. 383–411.

[21] P. LÖTSTEDT, *Numerical simulation of time-dependent contact and friction problems in rigid body mechanics*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 370–393.

[22] H. D. MITTELMANN, *On the efficient solution of nonlinear finite element equations* II, Numer. Math., 36 (1981), pp. 375–387.

[23] J. J. MORÉ AND G. TORALDO, *Algorithms for bound constrained quadratic programming problems*, Numer. Math., 55 (1989), pp. 377–400.

[24] D. P. O'LEARY, *A generalized conjugate gradient algorithm for solving a class of quadratic programming problems*, Linear Algebra Appl., 34 (1980), pp. 371–399.

[25] B. T. POLYAK, *The conjugate gradient method in extremal problems*, USSR Comput. Math. and Math. Phys., 9 (1969), pp. 94–112.

[26] S. J. WRIGHT, *Implementing proximal point methods for linear programming*, Report MCS-P45-0189, Argonne National Laboratory, Argonne, IL, 1989.

[27] E. K. YANG AND J. W. TOLLE, *A class of methods for solving large convex quadratic programs subject to box constraints*, preprint, Department of Operations Research, University of North Carolina, Chapel Hill, NC, 1988.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 114–122, February 1991

009

CONVERGENCE OF ITERATES OF AN INEXACT MATRIX SPLITTING ALGORITHM FOR THE SYMMETRIC MONOTONE LINEAR COMPLEMENTARITY PROBLEM*

O. L. MANGASARIAN†

Abstract. *Convergence of iterates* is established for a symmetric regular matrix splitting algorithm for the solution of the symmetric monotone linear complementarity problem where the subproblems are solved *inexactly*. The notable iterate convergence recently established by Luo and Tseng for *exact* subproblem solution is extended here to inexact subproblem solution for a symmetric matrix splitting. A principal application of the present result is to iterate convergence for the inexact block Jacobi method for which Pang and Yang established convergence of a *subsequence* of the iterates.

Key words. iterative matrix splitting, linear complementarity problems

AMS(MOS) subject classifications. 90C20, 15A39

1. Introduction. We consider the classical symmetric linear complementarity problem (LCP) of finding an x in the n-dimensional real space R^n such that

$$Mx + q \geq 0, \quad x \geq 0, \quad x(Mx + q) = 0, \tag{1.1}$$

where M is a given $n \times n$ real symmetric positive semidefinite (spsd) matrix and q is a given vector in R^n. This problem is equivalent to

$$\min_{x \geq 0} f(x) := \min_{x \geq 0} \frac{1}{2} xMx + qx. \tag{1.2}$$

Many iterative methods for solving this problem [1], [3], [5], [9]–[11], [13] can be modeled as follows. Split the matrix M [10] as follows:

$$M = B + C \tag{1.3}$$

and consider the sequence of (simpler) LCPs

$$Bx^{i+1} + Cx^i + q \geq 0, \quad x^{i+1} \geq 0, \quad x^{i+1}(Bx^{i+1} + Cx^i + q) = 0 \quad i = 0, 1, \cdots. \tag{1.4}$$

Convergence of a **subsequence** of the iterates $\{x^i\}$ for a variety of iterative methods [1], [3], [4], [5], [9]–[11], [13] can be established under the simple assumption of a **regular splitting**, that is,

$$M = B + C, \quad B - C \quad \text{positive definite.} \tag{1.5}$$

Recently Luo and Tseng [4] were the first to establish that the **whole** sequence $\{x^i\}$ generated by (1.4) converges for a regular splitting (1.5) for a spsd M when $f(x)$ is bounded below on the nonnegative orthant R^n_+. Their proof is rather complex and

*Received by the editors April 16, 1990; accepted for publication (in revised form) August 9, 1990. This material is based on research supported by National Science Foundation grants DCR-8521228 and CCR-8723091 and Air Force Office of Scientific Research grant AFOSR 89-0410.

†Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706 (olvi@cs.wisc.edu).

requires that the subproblems (1.4) be solved exactly. By contrast, our Algorithm 2.1 below requires only the approximate solution of (1.4) in a precisely defined and implementable way. However, our proof, which is considerably shorter, requires that B be symmetric. It is unclear whether the symmetry of B is the inevitable price one has to pay in order to allow inexactness in the solution of the subproblems (1.4). This was also the case in [13], where convergence is established for a subsequence of the iterates of a two-stage procedure for solving the symmetric LCP and where the inner iteration constituted an approximate solution of (1.4), with a symmetric B. An open question therefore remains: Can the symmetry assumption on B be removed from our principal result, Theorem 2.4, while maintaining inexactness of the subproblem solution?

We discuss our notation now. For a vector x in the n-dimensional real space R^n, x_+ will denote the vector in R^n with components $(x_+)_i := max\{x_i, 0\}$, $i = 1, \cdots, n$. A symmetric positive definite $n \times n$ real matrix induces an elliptic norm $\|\cdot\|_B$ on R^n, defined by $(xBx)^{1/2}$ for x in R^n. When $B = I$, we have the Euclidean or 2-norm $(xx)^{1/2}$, which we denote simply as $\|\cdot\|$. The one-norm of x, $\sum_{i=1}^n |x_i|$ will be denoted by $\|\cdot\|_1$. For an $m \times n$ real matrix A signified by $A \in R^{m\times n}$, A_i denotes the ith row, while A' denotes the transpose. A vector of ones in a real space of any dimension will be denoted by e without a superscript. The identity matrix of any order will be denoted by I. The nonnegative orthant in R^n will be denoted by R^n_+. The projection of a point x in R^n on a closed convex S set in R^n employing the norm $\|\cdot\|_B$ is defined as

$$arg \min_{p\in S} (p-x)B(p-x)$$

and is denoted by $p(x)$.

2. Iterate convergence of a symmetric matrix splitting. Before stating our algorithm, it is useful to note that the linear complementarity problem (1.1) with any matrix M is equivalent to

$$x = \big(x - (Mx + q)\big)_+. \tag{2.1}$$

This equivalence can be easily checked componentwise. Hence the subproblems (1.4) are equivalent to

$$x^{i+1} = \big(x^{i+1} - (Bx^{i+1} + Cx^i + q)\big)_+. \tag{2.2}$$

We shall assume that the associated quadratic function f is bounded below and hence the LCP (1.1) is solvable. Let X^* denote its closed convex solution set. Let $X^* \cap X_\alpha \neq \phi$, where for a positive number α,

$$X_\alpha := \{x | x \in R^n_+,\ ex \leq \alpha\}. \tag{2.3}$$

We are now ready to state our algorithm.

ALGORITHM 2.1. Given x^i determine x^{i+1} such that for some "error" sequence $\{h^i\} \subset R^n$ satisfying $\sum_{i=1}^\infty \|h^i\| < \infty$:

$$x^{i+1} = \big(h^{i+1} + x^{i+1} - (Bx^{i+1} + Cx^i + q)\big)_+, \quad i = 0, 1, \cdots, \tag{2.4a}$$

where $B + C$ is a regular splitting of M with $B' = B$ and

$$\sum_{i=0}^{\infty} \|h^{i+1}\| \cdot (\|x^{i+1} - x^i\| + \|x^i - p^i\|) < \infty, \tag{2.5a}$$

and $p^i := p(x^i)$ is the projection of x^i on $X^* \cap X_\alpha$ using the norm $\|\cdot\|_B$.

Note first that (2.4a), which is equivalent to the LCP

$$(2.4b)\quad Bx^{i+1} + Cx^i + q - h^{i+1} \geq 0, \quad x^{i+1} \geq 0, \quad x^{i+1}(Bx^{i+1} + Cx^i + q - h^{i+1}) = 0,$$

is solvable for all values of x^i and h^{i+1} in R^n because B is positive definite, which is in the class of matrices Q for which the linear complementarity problem is solvable for all values of problem data.

Before establishing the convergence of the iterates generated by Algorithm 2.1 we make a few remarks. The assumption that $X^* \cap X_\alpha \neq \phi$ does not imply the boundedness of X^* but merely that its intersection with the simplex X_α is nonempty. The positive number α is some upper bound on the 1-norm of a solution to the LCP (1.1) with least 1-norm. In general α is unknown, but is chosen sufficiently large to insure that $X^* \cap X_\alpha \neq \phi$. If α is not chosen large enough and nonconvergence of the iterates $\{x^i\}$ to a solution of the LCP (1.1) occurs, then this is easily detected and α can be increased. Note also that if there exists an $\hat{x} \geq 0$ which is not a solution of the LCP such that $M\hat{x} + q > 0$, then [6, Thm. 2.2] X^* is indeed bounded and α may be taken as:

$$(2.6)\qquad \alpha = \hat{x}(M\hat{x} + q) \Big/ \min_{1 \leq k \leq n} (M_k \hat{x} + q_k).$$

However, we do not assume the existence of such an $\hat{x}$. The size of α enters Algorithm 2.1 only in ensuring that condition (2.5a) is satisfied. This is discussed further in Remark 2.5 below. Lemma 2.6 below gives a precise way for implementing (2.5a). The plausibility of (2.5a) can be demonstrated as follows. Since B is positive definite, the subproblem (2.4a) is solvable for all values of h^{i+1}. Denote the explicit dependence of x^{i+1} on h^{i+1} by writing $x^{i+1}(h^{i+1})$. By [7, Thm. 3.3], $x^{i+1}(h^{i+1})$ is Lipschitzian in h^{i+1} with a Lipschitz constant ν depending on B only. Hence

$$\|h^{i+1}\| \cdot \|x^{i+1}(h^{i+1}) - x^i\| \leq \nu \|h^{i+1}\|^2 + \|h^{i+1}\| \cdot \|x^{i+1}(0) - x^i\|,$$

which ensures the smallness of the first term of (2.5a) by picking $\|h^{i+1}\|$ sufficiently small. The existence of an upper bound on $\|x^i - p^i\|$ in terms of x^i [8, Thm. 2.11] ensures the smallness of the second term of (2.5a). See Remark 2.5 and Lemma 2.6 for details.

We shall need the following simple but useful two lemmas, the first due to Cheng [1], which is a special case of a more general lemma [14, Lem. 2, p. 44].

LEMMA 2.2. [1, *Lem.* 2.1]. *Let* $\{e^i\}$ *and* $\{\varepsilon^i\}$ *be two sequences of nonnegative real numbers with* $\sum_{i=0}^{\infty} \varepsilon^i < \infty$ *and* $0 \leq e^{i+1} \leq e^i + \varepsilon^i$ *for* $i = 0, 1, \cdots$. *Then the sequence* $\{e^i\}$ *converges.*

LEMMA 2.3. [5, *Lem.* 2.2]. *For points* $x \in R^n$ *and* $y \in R^n_+$ *it follows that* $(x - x_+)(y - x_+) \leq 0$.

We are ready now to state and prove our principal convergence result. We remark that our proof is motivated by Polyak's convergence proof of the gradient projection algorithm [14, pp. 207–208]. However, a number of new ideas were needed such as introducing the inexactness h^{i+1} and the manner in which it is introduced and decreased, introducing the truncation X_α and projecting on its intersection with the solution set, and the use of the matrix B in the projection norm.

THEOREM 2.4. *Let the LCP* (1.1) *be solvable for some symmetric positive semidefinite* M. *Then the iterates* $\{x^i\}$ *of Algorithm* 2.1 *converge to a solution* x^* *of the LCP* (1.1).

Proof. By (2.4a) x^{i+1} is a projection on R^n_+ and hence it follows by Lemma 2.3 above that

$$\big(h^{i+1} - (Bx^{i+1} + Cx^i + q)\big)(p^i - x^{i+1}) \leq 0.$$

Since $(p^i - x^{i+1})(Mp^i + q) \leq 0$ it follows that

$$(p^i - x^{i+1})\big(B(x^i - x^{i+1}) - M(x^i - p^i)\big) \leq h^{i+1}(x^{i+1} - p^i). \tag{2.7}$$

From the identity

$$0 = \|x^i - p^i\|_B^2 - \|(x^i - x^{i+1}) - (p^i - x^{i+1})\|_B^2$$

and the symmetry of B we have

$$2(x^i - x^{i+1})B(p^i - x^{i+1}) = -\|x^i - p^i\|_B^2 + \|x^i - x^{i+1}\|_B^2 + \|x^{i+1} - p^i\|_B^2. \tag{2.8}$$

From the symmetry and positive semidefiniteness of M we have for $a,\ b \in R^n$

$$aMa + bMa \geq -bMb/4.$$

Hence

$$\begin{aligned}(x^{i+1} - p^i)M(x^i - p^i) &= (x^{i+1} - x^i)M(x^i - p^i) + (x^i - p^i)M(x^i - p^i)\\ &\geq -(x^{i+1} - x^i)M(x^{i+1} - x^i)/4.\end{aligned} \tag{2.9}$$

Using (2.8) and (2.9) in inequality (2.7) multiplied by 2, and invoking the positive definiteness of $B - C$ gives

$$\|x^{i+1} - p^i\|_B^2 \leq \|x^i - p^i\|_B^2 + 2h^{i+1}(x^{i+1} - p^i); \tag{2.10}$$

or equivalently (adding and subtracting p^{i+1} within the first term)

$$\begin{aligned}\|x^{i+1} - p^{i+1}\|_B^2 + 2(x^{i+1} - p^{i+1})B(p^{i+1} - p^i)\\ + \|p^{i+1} - p^i\|_B^2 \leq \|x^i - p^i\|_B^2 + 2h^{i+1}(x^{i+1} - p^i).\end{aligned} \tag{2.11}$$

Since

$$p^{i+1} = arg \min_{X^* \cap X_\alpha} (p - x^{i+1})\frac{B}{2}(p - x^{i+1}),$$

it follows by the Minimum Principle that

$$(p - p^{i+1})B(p^{i+1} - x^{i+1}) \geq 0 \quad \forall p \in X^* \cap X_\alpha.$$

Hence the second term in (2.11) is nonnegative and can be dropped. Dropping also the third term in (2.11), which is also nonnegative, gives

$$\|x^{i+1} - p^{i+1}\|_B^2 \leq \|x^i - p^i\|_B^2 + 2\|h^{i+1}\|(\|x^{i+1} - x^i\| + \|x^i - p^i\|).$$

It follows from (2.5a) of Algorithm 2.1 and Cheng's lemma that the sequence $\{\|x^i - p^i\|_B\}$ converges, and so does $\{\|x^i - p^i\|\}$ converge to β, say. Then for any $\delta > 0$ we have

$$-\delta \leq \|x^i - p^i\| - \beta \leq \delta \quad \forall i \geq i(\delta)$$

and hence

$$\|x^i\| \leq \|p^i\| + \beta + \delta \quad \forall i \geq i(\delta).$$

Since $\{p^i\} \subset X_\alpha$, it follows that $\{p^i\}$ is bounded and so is $\{x^i\}$. Now

$$\begin{aligned} f(x^i) - f(x^{i+1}) &= -(Mx^i + q)(x^{i+1} - x^i) - \|x^{i+1} - x^i\|^2_{M/2} \\ &= \left(x^{i+1} + h^{i+1} - \left[h^{i+1} + x^{i+1} - B(x^{i+1} - x^i) - (Mx^i + q)\right]\right) \cdot \\ &\quad (x^i - x^{i+1}) + \|x^{i+1} - x^i\|^2_{\frac{B-C}{2}} \\ &\geq h^{i+1}(x^i - x^{i+1}) + \|x^{i+1} - x^i\|^2_{\frac{B-C}{2}} \end{aligned}$$

(by Lemma 2.3, because x^{i+1} is the projection on R^n_+ of the term in the square bracket)

$$\geq -\|h^{i+1}\| \cdot \|x^{i+1} - x^i\| + \gamma \|x^{i+1} - x^i\|^2$$

(where γ is the smallest eigenvalue of $\frac{B-C}{2}$).

Hence

$$f(x^i) - f(x^{i+1}) \geq \gamma \|x^{i+1} - x^i\|^2 - \|h^{i+1}\| \cdot \|x^{i+1} - x^i\|. \tag{2.12}$$

Let $\bar{x} \in X^*$, then

$$0 \leq f(x^{i+1}) - f(\bar{x}) \leq \gamma \|x^{i+1} - x^i\|^2 + f(x^{i+1}) - f(\bar{x}) \leq f(x^i) - f(\bar{x}) + \|h^{i+1}\| \cdot \|x^{i+1} - x^i\|.$$

By (2.5a) we have that $\sum_{i=0}^{\infty} \|h^{i+1}\| \cdot \|x^{i+1} - x^i\| < \infty$ and $\{\|h^{i+1}\| \cdot \|x^{i+1} - x^i\|\} \to 0$. Hence, again by Cheng's lemma, the sequence $\{f(x^i) - f(\bar{x})\}$ converges, and so does the sequence $\{f(x^i)\}$. It follows from (2.12) that

$$0 = \lim_{i \to \infty} \left(f(x^i) - f(x^{i+1}) + \|h^{i+1}\| \cdot \|x^{i+1} - x^i\|\right) \geq \gamma \lim_{i \to \infty} \|x^{i+1} - x^i\|^2 \geq 0.$$

Hence $lim_{i \to \infty} \|x^{i+1} - x^i\| = 0$. Now, since B is positive definite, the single-valued map $x^{i+1} = T(x^i, h^{i+1})$ defined by (2.4a) or equivalently by (2.4b) is Lipschitzian [7, Thm. 3.3], with Lipschitz constant dependent on B only. Thus $lim_{i \to \infty} \|T(x^i, h^{i+1}) - x^i\| = 0$. Since $\{h^{i+1}\} \to 0$ and T is continuous, it follows that for an accumulation point x^* of the bounded sequence $\{x^i\}$, that $\{x^{i_j}\} \to x^*$ and $T(x^*, 0) = x^*$. The condition $T(x^*, 0) = x^*$ is equivalent to $x^* \in X^*$.

We now repeat the argument which begins this proof until we reach (2.10) but with x^* replacing p^i. Thus replacing p^i by x^* in (2.10) gives

$$\|x^{i+1} - x^*\|^2_B \leq \|x^i - x^*\|^2_B + 2\|h^{i+1}\| \cdot \|x^{i+1} - x^*\|. \tag{2.13}$$

Now employing the Lipschitz continuity of the projection operator p with Lipschitz constant μ and the fact that p^{i+1} is in X_α we have that

$$\begin{aligned} \|x^{i+1} - x^*\| &= \|x^{i+1} - x^i - p^{i+1} + p^i + x^i - p^i + p^{i+1} - x^*\| \\ &\leq (1 + \mu)\|x^{i+1} - x^i\| + \|x^i - p^i\| + \alpha + \|x^*\|. \end{aligned} \tag{2.14}$$

We note that μ may be taken as the ratio of the largest to the smallest eigenvalues of B. It follows then from (2.5a), from $\sum_{i=0}^{\infty} \|h^{i+1}\| < \infty$, and (2.14) that

$$\sum_{i=0}^{\infty} \|h^{i+1}\| \cdot \|x^{i+1} - x^*\| < \infty. \tag{2.15}$$

Hence by (2.13), (2.15), and Cheng's lemma we have that the sequence $\{\|x^i - x^*\|\}$ converges. We claim now that if $\{\|x^i - x^*\|\}$ converges to a positive number δ, say, a contradiction ensues. For

$$\frac{\delta}{2} > \|x^i - x^*\| - \delta > -\frac{\delta}{2} \qquad \forall i \geq \bar{i} \text{ for some } \bar{i}.$$

But since $\{x^{i_j}\} \to x^*$

$$\frac{\delta}{2} > \|x^{i_j} - x^*\| \quad \text{for some } i_j \geq \bar{i}.$$

The last two inequalities are contradictory. Hence $\|x^i - x^*\| \to 0$ and $lim_{i\to\infty} x^i = x^* \in X^*$. □

We discuss now how the inexactness condition (2.5a) of Algorithm 2.1 can be implemented precisely.

Remark 2.5. Since M is positive semidefinite we can, by a slight modification of [8, Thm. 2.7] replace $\|x^i - p^i\|$ in (2.5a) of Algorithm 2.1 by a computable error bound multiplied by a constant $\mu(M, B, q, \alpha)$ as follows:

$$\|x^i - p^i\| \leq \mu(M, B, q, \alpha)\Big[\|\big(x^i(Mx^i + q),\ -Mx^i - q,\ ex^i - \alpha\big)_+\| + \Big(\big(x^i(Mx^i + q)\big)_+ + \|(-Mx^i - q)_+\|\Big)^{1/2}\Big] = \mu(M, B, q, \alpha)\sigma(x^i),$$

where $\sigma(x^i)$ is defined by the term in the square bracket. Condition (2.5a) is then implied by

$$\sum_{i=0}^{\infty} \|h^{i+1}\| \cdot \big(\|x^{i+1} - x^i\| + \sigma(x^i)\big) < \infty. \tag{2.5b}$$

We give now a precise way of implementing (2.5b).

LEMMA 2.6. *Inequality* (2.5b) *and hence* (2.5a) *hold by taking* $\|h^{i+1}\|$ *equal to the largest element of* $\{\|h^i\|/2,\ \|h^i\|/4 \cdots\}$ *such that*

$$\|h^{i+1}\|\big(\|x^{i+1} - x^i\| + \sigma(x^i)\big) \leq \frac{\|h^i\|}{2}\big(\|x^i - x^{i-1}\| + \sigma(x^{i-1})\big). \tag{2.5c}$$

Proof. All we need to show is that (2.5c) holds for $\|h^{i+1}\|$ sufficiently small. Since B is positive definite we have that $x^{i+1} = x^{i+1}(h^{i+1})$ is Lipschitzian [7, Thm. 3.3] with constant ν depending on B only. Hence

$$\|x^{i+1}(h^{i+1}) - x^{i+1}(0)\| \leq \nu\|h^{i+1}\|.$$

Hence (2.5c) is implied by

$$\|h^{i+1}\|\big(\nu\|h^{i+1}\| + \|x^{i+1}(0) - x^i\| + \sigma(x^i)\big) \leq \frac{\|h^i\|}{2}\big(\|x^i - x^{i-1}\| + \sigma(x^{i-1})\big),$$

that is,

$$\text{(2.5d)}\quad \nu\|h^{i+1}\|^2+\|h^{i+1}\|\left[\|x^{i+1}(0)-x^i\|+\sigma(x^i)\right]-\left[\frac{\|h^i\|}{2}\left(\|x^i-x^{i-1}\|+\sigma(x^{i-1})\right)\right]\leq 0.$$

Defining the terms in the first and second square brackets in (2.5d) by ν^i and μ^i respectively, we have that (2.5d) is satisfied, and hence also (2.5a), if we take $\|h^{i+1}\| \in [0,\ \rho^{i+1}]$, where

$$\rho^{i+1}=\frac{-\nu^i+\sqrt{\nu^{i2}+4\nu\mu^i}}{2\nu}.$$

□

Remark 2.7. We note here that the sequence $\{x^i\}$ was determined as a function of the error sequence $\{h^i\}$ by solving the subproblems (2.4a) of Algorithm 2.1. This entails, then, an exact solution of the equivalent linear complementarity problem (2.4b) with a prescribed error term h^{i+1}, and in a certain sense that is at cross purposes to solving the original subproblems (1.4) inexactly. To avoid this we outline here a procedure that does not require exact subproblem solution. Let $\{y^i\}$ be a sequence of points in R^n which are obtained in any way as approximate solutions to the subproblems (1.4) with y replacing x in (1.4). We show now how the error sequence $\{h^i\}$ satisfying (2.4a) is computed from $\{y^i\}$. For this purpose we first define the **computable** error bound in satisfying (1.4) as follows. Let

$$\text{(2.16a)}\qquad e^{i+1}=e^{i+1}(y^{i+1}):=min\,\{y^{i+1},\ By^{i+1}+Cy^i+q\},$$

or equivalently,

$$\text{(2.16b)}\qquad e^{i+1}:=y^{i+1}-\left(y^{i+1}-(By^{i+1}+Cy^i+q)\right)_+.$$

By [12, Lem. 2], the error $\|y^{i+1}-y^{i+1}(0)\|$, where $y^{i+1}(0)$ is the unique solution of (1.4) with y replacing x, is bounded by the computable e^{i+1} as follows:

$$\text{(2.17)}\qquad \|y^{i+1}-y^{i+1}(0)\|\leq\lambda(B)\|e^{i+1}\|,$$

where

$$\text{(2.18)}\qquad \lambda(B):=1+\frac{\|I-B\|}{\alpha},\quad \alpha:=\text{min eigenvalue }(B)>0.$$

Note that e^{i+1} is easily computable from (2.16a) and hence can be used as a simple measure of how exactly y^{i+1} satisfies (1.4). We now relate h^{i+1} to e^{i+1}. From (2.16b) we have that y^{i+1} solves the linear complementarity problem

$$\text{(2.19a)}\quad w^{i+1}=By^{i+1}+Cy^i+q-e^{i+1}\geq 0,\quad y^{i+1}-e^{i+1}\geq 0,\quad (y^{i+1}-e^{i+1})w^{i+1}=0,$$

or equivalently,

$$\text{(2.19b)}\qquad \begin{gathered} w^{i+1}=B(y^{i+1}-e^{i+1})+C(y^i-e^i)+q+(B-I)e^{i+1}+Ce^i\geq 0,\\ y^{i+1}-e^{i+1}\geq 0,\quad (y^{i+1}-e^{i+1})w^{i+1}=0.\end{gathered}$$

By defining

$$\text{(2.20)}\qquad x^{i+1}:=y^{i+1}-e^{i+1},\quad x^i=y^i-e^i,\quad h^{i+1}:=(I-B)e^{i+1}-Ce^i,$$

the subproblem (2.19b) reduces to (2.4b), and the error term h^{i+1} can be computed from the relation $h^{i+1} = (I - B)e^{i+1} - Ce^i$ in (2.20). Thus the smallness condition (2.5b) on the sequence $\{\|h^i\|\}$ can be translated, through the relations (2.20), (2.16b) and the nonexpansiveness of the plus function $(\cdot)_+$, into a smallness condition on the sequence $\{\|e^i\|\}$ as follows:

$$\sum_{i=0}^{\infty} (\|e^{i+1}\| + \|e^i\|)\big(\|y^{i+1} - y^i\| + \|y^i - y^{i-1}\| + \sigma(y^i - e^i)\big) < \infty. \tag{2.21}$$

3. Conclusions. We have established convergence of the iterates for a symmetric regular splitting algorithm for the symmetric monotone linear complementarity problem. The principal application is probably to an inexact block Jacobi method for solving the symmetric LCP. In particular, if we let

$$M = L + D + L', \tag{3.1}$$

where D is some **block** diagonal of M and $L + L'$ is the remaining part of M, then we can take

$$B = \lambda I + D, \quad C = -\lambda I + L + L', \quad B - C = 2(\lambda I + D) - M. \tag{3.2}$$

This splitting is regular for

$$\lambda > \max \text{ eigenvalue } \Big(\frac{M}{2} - D\Big). \tag{3.3}$$

The splitting (3.2) is useful in the parallel solution of linear programs where the constraints of the problem are distributed among the processors and the objective function is appropriately modified for each processor by Lagrangian and proximal terms. This will be discussed in a forthcoming paper [2].

REFERENCES

[1] Y. C. Cheng, *On the gradient-projection method for solving the nonsymmetric linear complementarity problem*, J. Optim. Theory Appl.,43 (1984), pp. 527–541.

[2] M. C. Ferris and O. L. Mangasarian, *Parallel constraint distribution*, Tech. Report 971, Computer Sciences Department, University of Wisconsin, Madison, WI, 1990.

[3] Y. Y. Lin and J.-S. Pang, *Iterative methods for large convex quadratic programs: A survey*, SIAM J. Control Optim., 25 (1987), pp. 383–411.

[4] Z.-Q. Luo and P. Tseng, *On the convergence of a matrix splitting algorithm for the symmetric linear complementarity problem*, SIAM J. Control Optim., 29(1991), to appear.

[5] O. L. Mangasarian, *Solution of symmetric linear complementarity problems by iterative methods*, J. Optim. Theory Appl., 22 (1977), pp. 465–485.

[6] ———, *Simple computable founds for solutions of linear complementarity problems and linear programs*, Math. Programming Stud., 25 (1985), pp. 1–12.

[7] O. L. Mangasarian and T.H. Shiau, *Lipschitz continuity of solutions of linear inequalities, programs, and complementarity problems*, SIAM J. Control Optim., 25 (1987), pp. 583–595.

[8] ———, *Error bounds for monotone linear complementarity problems*, Math. Programming, 36 (1986), pp. 81–89.

[9] J.-S. Pang, *On the convergence of a basic iterative method for the implicit complementarity problem*, J. Optim. Theory Appl., 37 (1982), pp. 149–162.

[10] ———, *Necessary and sufficient conditions for the convergence of iterative methods for the linear complementarity problem*, J. Optim. Theory Appl., 42 (1984), pp. 1–17.

[11] J.-S. PANG, *More results on the convergence of iterative methods for the symmetric linear complementarity problems*, J. Optim. Theory Appl., 49 (1986), pp. 107–134.
[12] ———, *Inexact Newton methods for the nonlinear complementarity problem*, Math. Programming, 36 (1986), pp. 54–71.
[13] J.-S. PANG AND J.-M. YANG, *Two-stage parallel iterative methods for the symmetric linear complementarity problem*, Ann. Oper. Res., 14 (1988), pp. 61–75.
[14] B. T. POLYAK, *Introduction to Optimization*, Optimization Software, Inc., New York, 1987.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 123–145, February 1991

010

ON THE CONVERGENCE OF THE MULTIDIRECTIONAL SEARCH ALGORITHM*

VIRGINIA TORCZON†

Abstract. This paper presents the convergence analysis for the multidirectional search algorithm, a direct search method for unconstrained minimization. The analysis follows the classic lines of proofs of convergence for gradient-related methods. The novelty of the argument lies in the fact that explicit calculation of the gradient is unnecessary, although it is assumed that the function is continuously differentiable over some subset of the domain. The proof can be extended to treat most nonsmooth cases of interest; the argument breaks down only at points where the derivative exists but is not continuous. Finally, it is shown how a general convergence theory can be developed for an entire class of direct search methods—which includes such methods as the factorial design algorithm and the pattern search algorithm—that share a key feature of the multidirectional search algorithm.

Key words. unconstrained optimization, convergence analysis, direct search methods, parallel optimization, multidirectional search, Nelder–Mead simplex algorithm

AMS(MOS) subject classifications. 49D30, 65K05

1. Introduction.

In this paper we shall give the convergence analysis for the multidirectional search algorithm [13]. The multidirectional search algorithm is a direct search method designed to solve the unconstrained minimization problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}),$$

where $f : \mathbb{R}^n \to \mathbb{R}$.

Direct search algorithms presume little of the function—typically only that the function is continuous—since they do not require, or even directly estimate, gradient information. It has long been recognized that these methods are gradient-related, but the convergence analysis for a large class of these algorithms has been incomplete, as we shall discuss in the next section.

Our proof of convergence for the multidirectional search algorithm will follow the classic lines of proofs for gradient-related methods. First we will show that the multidirectional search algorithm is a descent method. Then we will show that the search directions do not deteriorate. Finally we will show that the algorithm satisfies a notion of sufficient decrease in the value of the objective function for the size of the step taken. While we assume that the function is continuously differentiable over some subset of the domain, we never explicitly compute the gradient. The novelty in this argument lies in the fact that the algorithm provides enough structure to make explicit information about the gradient unnecessary. The multidirectional search algorithm will be introduced in §3. The convergence analysis for the differentiable case will be developed in §§4, 5, and 6.

The result for the smooth case can also be extended to handle most nonsmooth cases of interest. This extension will be given in §7. Finally, we will close by reviewing both what the convergence analysis tells us about how the multidirectional search algorithm works and what it also suggests about a more general convergence theory for an entire class of direct search algorithms, which is a subject of our current research.

* Received by the editors March 19, 1990; accepted for publication (in revised form) June 12, 1990. This research was sponsored by SDIO/IST/ARO and the Air Force Office of Scientific Research grant 89-0363. A portion of this work is contained in the author's doctoral thesis under the supervision of J. E. Dennis, Jr. in the Department of Mathematical Sciences, Rice University, Houston, Texas 77251-1892.

† Department of Mathematical Sciences, Rice University, Houston, Texas 77251-1892.

2. Direct search methods. Direct search methods are characterized by the fact that they do not use derivatives. They forgo this very useful information for very practical reasons. Often, particularly in experimental settings, analytic derivatives are simply unavailable. Finite-difference approximations to the gradient could be used, but the cost of computing the function values on a sequential machine may make this option prohibitively expensive. Furthermore, in an experimental setting it is not at all unusual to have function values that can only be trusted to a few significant digits so that finite-difference approximations to the gradient may prove unreliable. Another possibility is that the experimental apparatus itself may make finite-difference approximations to the gradient difficult, if not impossible.

Having relinquished explicit derivative information, the direct search methods typically consider a natural alternative: at every iteration they explore each direction in a linearly independent set of n search directions. As we shall see, if this set of search directions has the right structure, it is possible to derive convergence results for these algorithms. Our interest in direct search methods arose from the observation that at every iteration the algorithms typically perform searches in each of n directions.

Thus, it certainly seems reasonable to assume that in most, if not all, of these algorithms the n searches required for a single iteration can be conducted independently. This, in turn, suggests a natural way to develop new optimization algorithms for parallel machines [5], [13].

The direct search algorithms are distinguished both by the way in which the set of n search directions is chosen and maintained and by the way "exploratory" steps are taken in each of the n directions. The most important distinction, for theoretical purposes, is between those methods for which the set of search directions is modified at the end of each iteration and those methods for which the set of search directions remains fixed across all iterations. Examples of algorithms in the first class include Rosenbrock's method [11], Powell's method [10], and Zangwill's method [16]. In the second class of direct search methods one finds such examples as the factorial design algorithm of Box [2], the pattern search algorithm of Hooke and Jeeves [7], and the simplex algorithm of Spendley, Hext, and Himsworth [12] (the method upon which the simplex algorithm of Nelder and Mead is based).

The Nelder–Mead simplex algorithm [8], perhaps the most popular of the direct search methods, does not search in each of n linearly independent search directions at every iteration. It is worth noting that while we know of several attempts to prove convergence of the Nelder–Mead simplex algorithm, none of these has met with success. Furthermore, the experimental evidence gathered while testing and comparing the multidirectional search algorithm with the Nelder–Mead simplex algorithm suggests that a convergence result for the Nelder–Mead simplex algorithm may not be possible [13]. The fact that the Nelder–Mead simplex algorithm only searches in a single direction at each iteration removes it from the domain of this analysis—an observation which first suggested how the Nelder–Mead simplex algorithm might fail. However, as yet we can offer no satisfactory explanation of *why* this failure occurs. (For a further discussion of the experimental results, see [13].)

The methods which modify the set of search directions at the end of each iteration use the result of the exploratory searches along each of n linearly independent search directions to compute a new search direction—that defined by the point used to start the iteration and the point found at the conclusion of the n exploratory steps. The methods are distinguished by the way in which the new set of search directions is then determined. As Zangwill [16] points out, care must be taken to ensure that the set

of search directions remains linearly independent. However, once proper precautions are observed, several nice results can be derived, as shown by both Powell [10] and Zangwill [16] in their original papers, as well as those found in later works. (See [1], [9].)

The direct search methods for which the search directions remain fixed across all iterations have been analyzed with varying degrees of success. One of the best known algorithms of this sort is the pattern search algorithm of Hooke and Jeeves. Céa [3] gives a concise proof of convergence that uses few of the features of the pattern search algorithm. His analysis does, however, rely on the fact that the algorithm requires the step sizes to be monotonically decreasing. The multidirectional search algorithm allows the steps to increase in size if there is a corresponding decrease in the value of the objective function. Thus, Céa's result cannot be extended to the multidirectional search algorithm. Until recently, this has been the only convergence result of which we were aware for this class of direct search methods. We have since discovered a more ambitious convergence analysis undertaken by Yu Wen-ci [14], [15]. He, too, noted that the pattern search algorithm of Hooke and Jeeves and the original simplex algorithm of Spendley, Hext, and Himsworth share the features of fixed search directions and fixed rescaling factors. Using these observations, and the notion of a positive basis, he derived a general convergence result for modified versions of both these algorithms.

Our analysis differs from that of Yu Wen-ci in two important respects. First, Yu Wen-ci, like Céa, must assume that the step sizes are monotonically decreasing. Again, this means that his analysis cannot be extended to cover the multidirectional search algorithm. Our analysis allows the step sizes to increase if decreases are seen in the corresponding objective function values.

Second, to show that these algorithms cannot take steps that are too long relative to the amount of decrease in the value of the objective function— in other words, that there is *sufficient* decrease in the value of the objective function for the size of the step taken—Yu Wen-ci introduces the notion of an "error-controlling" sequence. Unfortunately, it is not clear how such a sequence can be constructed and maintained in practice. In fact, one of the difficulties in applying the classical analysis for descent methods to these algorithms is that an explicit calculation of the initial descent slope is used to enforce the Armijo–Goldstein–Wolfe conditions for sufficient decrease (see [4], [9]). Direct search methods, by definition, do not compute such information. Our result places no such burden on the multidirectional search algorithm. We can guarantee that the algorithm cannot take steps that are too long, relative to the amount of decrease in the object function value, *without* enforcing the Armijo–Goldstein–Wolfe conditions or introducing any other such measure of sufficient decrease. The key feature of the multidirectional search algorithm which makes this analysis possible is that both the search directions and the scaling factors which determine the step sizes are fixed across all iterations of the algorithm. This feature can also be found in several of the traditional sequential direct search methods, so one of the goals of our current research is to extend the results presented here for the multidirectional search algorithm to these other direct search methods. The only limitation we must impose is that the scaling factors used to determine the size of the steps taken must be rational. This mild restriction is easily satisfied by all the algorithms in this class; the values typically recommended for the scaling factors are $\frac{1}{2}$ and 2.

We are now ready to introduce the multidirectional search algorithm.

3. Algorithm. A formal statement of the multidirectional search algorithm is given on the next page. We proceed with a brief description of the algorithm.

In order to understand the algorithm, consider the sequence of "best" vertices $\{\mathbf{v}_0^k\}$, where by "best" we mean that for all k,

$$f(\mathbf{v}_0^k) \leq f(\mathbf{v}_i^k) \quad \text{for } i = 1, \cdots, n.$$

At each iteration of the inner **repeat** loop a search is conducted from the current best vertex $\mathbf{v}_0^k$ in each of the n directions determined by the n edges adjacent to $\mathbf{v}_0^k$. The goal of the search is to replace $\mathbf{v}_0^k$, i.e., to find a new vertex with a function value that is strictly less than the function value at $\mathbf{v}_0^k$.

There are three possible trial steps: the rotation[1] step, the expansion step, and the contraction step, which are illustrated in Fig. 1. Note that the algorithm always computes the rotation step and then tests to see if a new best vertex has been found. If a new best vertex has been identified, an expansion step is computed. Otherwise, the algorithm computes, and automatically accepts, the contraction step.

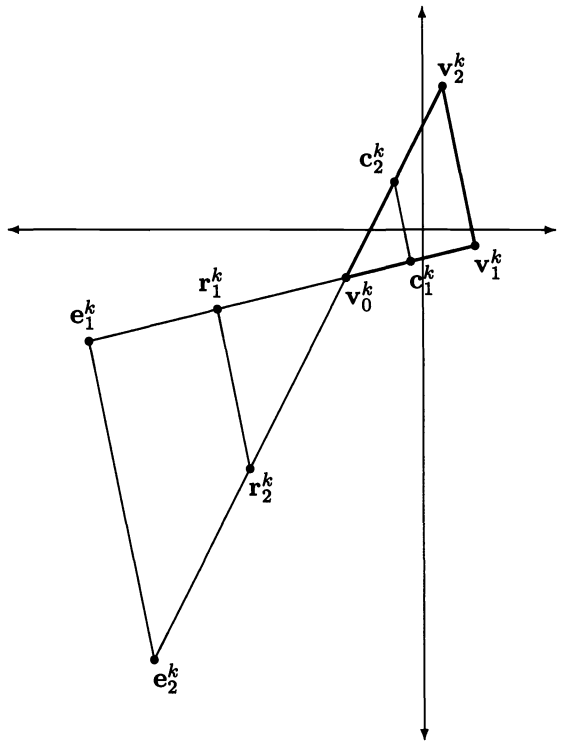


FIG. 1. *The three possible steps given the simplex S_k with vertices $\langle \mathbf{v}_0^k, \mathbf{v}_1^k, \mathbf{v}_2^k \rangle$.*

The algorithm can thus be viewed in terms of its two primary loops: the outer **while** loop, which determines a new set of search directions by considering a new best vertex, and the inner **repeat** loop, which determines the length of the steps to be taken.

[1] The "rotation" step is analogous to the "reflection" step found in the simplex algorithm of Nelder and Mead [8]. We choose to depart from the terminology used by Nelder and Mead since "rotation" more accurately describes the effect of the step on the position of the simplex.

THE MULTIDIRECTIONAL SEARCH ALGORITHM

Given an initial simplex S_0 with vertices $\langle \mathbf{v}_0^0, \mathbf{v}_1^0, \cdots, \mathbf{v}_n^0 \rangle$, choose $\mu, \theta \in Q$ such that
/* expansion factor */
$\mu \in (1, +\infty)$, and
/* contraction factor */
$\theta \in (0, 1)$.

for $i = 0, \cdots, n$ /*** initialization loop **/
  calculate $f(\mathbf{v}_i^k)$
end
$k \leftarrow 0$
while (stopping criterion is not satisfied) **do** /*** outer **while** loop **/
  /* find a new best vertex */
  $j \leftarrow \arg\min_i \{f(\mathbf{v}_i^k) : i = 0, \cdots, n\}$
  swap $\mathbf{v}_j^k$ and $\mathbf{v}_0^k$
  repeat /* inner **repeat** loop*/
    Check the stopping criterion.
    /* rotation step */
    for $i = 1, \cdots, n$
      $\mathbf{r}_i^k \leftarrow \mathbf{v}_0^k - (\mathbf{v}_i^k - \mathbf{v}_0^k)$
      calculate $f(\mathbf{r}_i^k)$
    end
    replaced $= (\min \{f(\mathbf{r}_i^k) : i = 1, \cdots, n\} < f(\mathbf{v}_0^k))$
    if *replaced* **then**
      /* expansion step */
      for $i = 1, \cdots, n$
        $\mathbf{e}_i^k \leftarrow \mathbf{v}_0^k - \mu (\mathbf{v}_i^k - \mathbf{v}_0^k)$
        calculate $f(\mathbf{e}_i^k)$
      end
      if $(\min \{f(\mathbf{e}_i^k) : i = 1, \cdots, n\} < \min \{f(\mathbf{r}_i^k) : i = 1, \cdots, n\})$ **then**
        /* accept expansion */
        $\mathbf{v}_i^k \leftarrow \mathbf{e}_i^k$ for $i = 1, \cdots, n$
      else
        /* accept rotation */
        $\mathbf{v}_i^k \leftarrow \mathbf{r}_i^k$ for $i = 1, \cdots, n$
      endif
    else
      /* contraction step */
      for $i = 1, \cdots, n$
        $\mathbf{c}_i^k \leftarrow \mathbf{v}_0^k + \theta (\mathbf{v}_i^k - \mathbf{v}_0^k)$
        calculate $f(\mathbf{c}_i^k)$
      end
      replaced $= (\min \{f(\mathbf{c}_i^k) : i = 1, \cdots, n\} < f(\mathbf{v}_0^k))$
      /* accept contraction */
      $\mathbf{v}_i^k \leftarrow \mathbf{c}_i^k$ for $i = 1, \cdots, n$
    endif
  until *replaced* /* end **repeat** loop */
  $k \leftarrow k + 1$
end /**** end **while** loop ***/

The expansion factor μ and the contraction factor θ determine the lengths, relative to the original edges in the simplex, of the steps to be considered. In our implementation, μ is set equal to 2 and θ is set equal to $\frac{1}{2}$, as in the example shown in Fig. 1. Further discussion of the choice of scaling factors, as well as additional implementation details, can be found in [13].

4. Key lemmas. Two important results follow directly from the description of the multidirectional search algorithm. First, the multidirectional search algorithm is a *descent method* since

$$f(\mathbf{v}_0^{k+1}) < f(\mathbf{v}_0^k).$$

For this to be true we need only show that if the best vertex $\mathbf{v}_0^k$ is at a point where the function is differentiable, and $\nabla f(\mathbf{v}_0^k)$ is not equal to zero, then the inner **repeat** loop terminates in a finite number of iterations. Second, the search directions determined by the multidirectional search algorithm are *uniformly linearly independent.* This means that there exists a constant $\gamma > 0$ such that for all $k \geq 0$ and $\mathbf{x} \neq 0$,

$$\max\left\{\frac{|\mathbf{x}^T(\mathbf{v}_i^k - \mathbf{v}_0^k)|}{\|\mathbf{x}\|\,\|\mathbf{v}_i^k - \mathbf{v}_0^k\|}, i = 1, \cdots, n\right\} \geq \gamma.$$

This follows from the observation that while the algorithm may translate and rotate the simplex, or change its scale, the angles in the original simplex S_0 are preserved across all iterations of the algorithm. Both results are given in the next two lemmas. We begin by introducing the following notation.

Let $\mathbf{v}_0^k$ be the best vertex of the simplex S_k used to start the kth iteration of the algorithm. Define the level set of f at $\mathbf{v}_0^k$ to be

$$L(\mathbf{v}_0^k) = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{v}_0^k)\}.$$

Given $\mathbf{y} \in \mathbb{R}^n$, let the contour $C(\mathbf{y})$ be

$$C(\mathbf{y}) = \{\mathbf{x} : f(\mathbf{x}) = f(\mathbf{y})\}.$$

Let X_* be the set of stationary points of the function f in $L(\mathbf{v}_0^0)$.

We will assume, for now, that f is continuously differentiable on $L(\mathbf{v}_0^0)$; however, as we will discuss in §7, if we redefine X_*, this assumption can be weakened. We next show that the multidirectional search algorithm is a descent method.

LEMMA 4.1. *Suppose that f is continuously differentiable on $L(\mathbf{v}_0^0)$. If the multidirectional search algorithm finds an iterate $\mathbf{v}_0^{k+1}$, then*

$$f(\mathbf{v}_0^{k+1}) < f(\mathbf{v}_0^k).$$

If $\nabla f(\mathbf{v}_0^k) \neq 0$, then the multidirectional search algorithm will find such a $\mathbf{v}_0^{k+1}$.

Proof. The multidirectional search algorithm will accept a new iterate $\mathbf{v}_0^{k+1}$ *only* if $f(\mathbf{v}_0^{k+1}) < f(\mathbf{v}_0^k)$, so the first conclusion in the lemma is tautologically true. The more interesting point is the second: if $\mathbf{v}_0^k$ is not a critical point—in particular, if $\nabla f(\mathbf{v}_0^k) \neq 0$—then the algorithm will find an acceptable $\mathbf{v}_0^{k+1}$ in a finite number of iterations of its inner **repeat** loop. This should not be too surprising since the algorithm is performing line searches along the n directions determined by the n edges adjacent to $\mathbf{v}_0^k$.

We will assume that $\mathbf{v}_0^k \notin X_*$ and show that the inner **repeat** loop terminates in a finite number of iterations.

The set of edges adjacent to any vertex in a simplex is linearly independent. Thus, the set of n edges adjacent to the current best vertex $\mathbf{v}_0^k$,

$$\left\{(\mathbf{v}_i^k - \mathbf{v}_0^k) : i = 1, \cdots, n\right\},$$

spans $I\!R^n$.

If $\mathbf{v}_0^k \notin X_*$, then $\nabla f(\mathbf{v}_0^k)$ is nonzero. Consequently, there exists at least one i, for $i = 1, \cdots, n$, such that

$$\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) \neq 0\,.$$

There are then two cases to consider.

Case 1. $\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) > 0$.

Note that since $(\mathbf{v}_i^k - \mathbf{v}_0^k) = -(\mathbf{r}_i^k - \mathbf{v}_0^k)$,

$$\nabla f(\mathbf{v}_0^k)^T(\mathbf{r}_i^k - \mathbf{v}_0^k) < 0\,.$$

Consider the right-hand directional derivative of f at $\mathbf{v}_0^k$ in the direction $(\mathbf{r}_i^k - \mathbf{v}_0^k)$:

$$f'(\mathbf{v}_0^k)^T(\mathbf{r}_i^k - \mathbf{v}_0^k) = \lim_{h \to 0^+} \frac{f\left(\mathbf{v}_0^k + h\,(\mathbf{r}_i^k - \mathbf{v}_0^k)\right) - f(\mathbf{v}_0^k)}{h} < 0\,.$$

Thus, there exists an $h_k > 0$ such that for $0 < t \leq h_k$

$$f\left(\mathbf{v}_0^k + t(\mathbf{r}_i^k - \mathbf{v}_0^k)\right) < f(\mathbf{v}_0^k)\,.$$

Case 2. $\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) < 0$.

Consider the right-hand directional derivative of f at $\mathbf{v}_0^k$ in the direction $(\mathbf{v}_i^k - \mathbf{v}_0^k)$:

$$f'(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) = \lim_{h \to 0^+} \frac{f\left(\mathbf{v}_0^k + h\,(\mathbf{v}_i^k - \mathbf{v}_0^k)\right) - f(\mathbf{v}_0^k)}{h} < 0\,.$$

Thus, there exists an $\bar{h}_k \in (0, h_k]$ such that for $0 < t \leq \bar{h}_k$

$$f\left(\mathbf{v}_0^k + t\,(\mathbf{v}_i^k - \mathbf{v}_0^k)\right) < f(\mathbf{v}_0^k)\,.$$

Conclusion. The vertices of the contracted simplex are defined to be

$$\mathbf{c}_i^k = \mathbf{v}_0^k + \theta\left(\mathbf{v}_i^k - \mathbf{v}_0^k\right),$$

for $i = 1, \cdots, n$, where $\theta \in (0, 1)$ is the fixed contraction factor. If the contracted step is accepted at the current iteration of the inner **repeat** loop, and a new best vertex has not been found, then at the next iteration of the inner loop, the rotated step can be defined in terms of the contracted vertices as

$$\begin{aligned}\mathbf{r}_i^k &= \mathbf{v}_0^k - \left(\mathbf{c}_i^k - \mathbf{v}_0^k\right) \\ &= \mathbf{v}_0^k - \left(\left(\mathbf{v}_0^k + \theta\left(\mathbf{v}_i^k - \mathbf{v}_0^k\right)\right) - \mathbf{v}_0^k\right) \\ &= \mathbf{v}_0^k - \theta(\mathbf{v}_i^k - \mathbf{v}_0^k)\,.\end{aligned}$$

Therefore, for any k, if $\mathbf{v}_0^k \notin X_*$, then there exists a positive integer p_k, such that $\theta^{p_k} < \bar{h}_k$ and so the inner **repeat** loop terminates in a finite number of iterations as required. □

LEMMA 4.2. *The search directions determined by the multidirectional search algorithm are uniformly linearly independent.*

Proof. Consider each of the three possible steps the multidirectional search algorithm may take.

1. The rotation step rotates the given initial simplex about the best vertex. This could be viewed as first translating the given initial simplex by $-\mathbf{v}_0^k$, applying the transformation $-I$, and then translating the simplex by $\mathbf{v}_0^k$. Since $-I$ is an orthogonal transformation, the Euclidean inner product is preserved. Thus, neither the translations nor the rotation affect the angles in the given initial simplex.
2. The expansion step rescales the rotated simplex by a factor of μ, which again leaves the angles of the given initial simplex unchanged.
3. The contraction simplex is formed by first translating the rotated simplex by $-\mathbf{v}_0^k$, applying the orthogonal transformation $-I$, and then translating the simplex by $\mathbf{v}_0^k$, which simply restores the initial simplex. The simplex is then rescaled by a factor of θ. Thus, the angles of the given initial simplex are unchanged.

Since none of the three possible steps affect the angles in the initial simplex, the angles of the simplex used to start the algorithm S_0 remain constant across all iterations of the multidirectional search algorithm. Therefore, there exists a $\gamma > 0$, which does not depend on k, such that

$$\max\left\{\frac{|\mathbf{x}^T(\mathbf{v}_i^k - \mathbf{v}_0^k)|}{\|\mathbf{x}\|\,\|\mathbf{v}_i^k - \mathbf{v}_0^k\|}, i = 1, \cdots, n\right\} \geq \gamma \tag{1}$$

for all $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{x} \neq 0$, as required. □

5. The smooth case. We are now ready for a formal statement of the theorem.

THEOREM 5.1. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuously differentiable on $L(\mathbf{v}_0^0)$. Then some subsequence of $\{\mathbf{v}_0^k\}$ converges to a point $\mathbf{x}_* \in X_*$. Furthermore, $\{\mathbf{v}_0^k\}$ converges to $C_* = C(\mathbf{x}_*)$ in the sense that*

$$\lim_{k\to\infty}\left[\inf_{\mathbf{x}\in C_*}\|\mathbf{v}_0^k - \mathbf{x}\|\right] = 0\,.$$

COROLLARY 5.2. *Assume that f is continuously differentiable, $L(\mathbf{v}_0^0)$ is compact, and f is strictly convex on $L(\mathbf{v}_0^0)$. Then*

$$\lim_{k\to\infty}\mathbf{v}_0^k = \mathbf{x}_*\,,$$

where $\mathbf{x}_*$ is the unique minimizer of f in $L(\mathbf{v}_0^0)$.

We will show that if the conclusion of Theorem 5.1 were not true, then there would be a lower bound on the lengths of the edges of the simplices generated by the algorithm. This, in turn, would imply that the algorithm can generate at most a finite number of iterates, which would contradict the fact that away from X_* the algorithm produces a sequence of points with strictly monotonically decreasing function values, as we showed in Lemma 4.1.

The proof of Theorem 5.1 will be given in §6. The rest of this section is devoted to results needed for the proof.

5.1. Enforcing step length control. Together, Lemmas 4.1 and 4.2 guarantee that if f is continuously differentiable on $L(\mathbf{v}_0^0)$, then the multidirectional search algorithm generates at least one direction of descent that is uniformly bounded away from orthogonality with the gradient. However, we still need to ensure that the steps

the algorithm takes are neither too long nor too short relative to the amount of decrease seen in the value of the objective function. In most other descent methods, step length control is guaranteed by enforcing the Armijo–Goldstein–Wolfe conditions. The multidirectional search algorithm cannot enforce the Armijo–Goldstein–Wolfe conditions because it does not have explicit knowledge of the gradient. There is enough structure in the algorithm, however, to enforce step length control without enforcing the Armijo–Goldstein–Wolfe conditions.

We begin by proving the existence of an upper bound on the lengths of the edges of the simplex.

PROPOSITION 5.3. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuously differentiable on $L(\mathbf{v}_0^0)$. Then there exists a constant $M > 0$ such that*

$$\|\mathbf{v}_i^k - \mathbf{v}_0^k\| \leq M \qquad \forall\, i, k\,.$$

Proof. Assume that $\mathbf{v}_0^0 \notin X_*$. If so, in the proof of Lemma 4.1 we showed that the multidirectional search algorithm will produce a new best vertex in a finite number of iterations of the inner **repeat** loop. Since $L(\mathbf{v}_0^0)$ is compact, for all $k \geq 1$ there exists at least one vertex $\mathbf{v}_i^k$ such that the vertices $\mathbf{v}_0^k, \mathbf{v}_i^k \in L(\mathbf{v}_0^0)$. Since $L(\mathbf{v}_0^0)$ is bounded, this implies a bound on the length of the edge $(\mathbf{v}_i^k - \mathbf{v}_0^k)$. Since the rescaling factors μ and θ are constant across all edges of the simplex for all iterations of the multidirectional search algorithm, the relative lengths of all edges in the simplex remain the same across all iterations of the algorithm. This implies the existence of $M > 0$ as required.

If $\mathbf{v}_0^0 \in X_*$, we can no longer argue that the inner **repeat** loop will produce a new best vertex in a finite number of iterations. There are then two possibilities. The first is that the multidirectional search algorithm produces a new best vertex. If so, the argument given above still holds. The other possibility is that the multidirectional search algorithm does not find a new best vertex, i.e., the inner **repeat** loop does not terminate. If so, then the contraction step has been accepted, because the rotation step and the expansion step are accepted only when they produce a new best vertex. Since the contraction factor θ is strictly less than one, this means that the length of every edge in the simplex is strictly monotonically decreasing. Thus, the maximum length across all edges in the initial simplex S_0 provides an upper bound on the length of all edges in all subsequent simplices, as required. □

The existence of an upper bound on the lengths of the edges of the simplex implies the existence of a compact set, which we shall call $\mathcal{M}$, that contains $L(\mathbf{v}_0^0)$ and all the simplices generated by the multidirectional search algorithm from a given initial simplex S_0.

We next posit the following *null* hypothesis: the sequence of best vertices $\mathbf{v}_0^k$ stays uniformly bounded away from X_*. We will show that under this hypothesis a lower bound on the lengths of the edges in the simplex exists and that once the edges in the simplex become small enough, the rotation step will always be acceptable. Note that this does not necessarily mean that the rotation step will be accepted. If the rotation step is acceptable, the multidirectional search algorithm automatically considers the expansion step; however, we are guaranteed that the contraction step will not be considered. Since the lengths of the edges in the simplex are reduced only when the simplex is contracted, this argument ensures no further reduction in the size of the simplex.

PROPOSITION 5.4. *Assume that $L(\mathbf{v}_0^0)$ is compact and f is continuously differentiable on $L(\mathbf{v}_0^0)$. Suppose that the best vertices stay uniformly bounded away from*

X_*; i.e., for a fixed $\epsilon > 0$, independent of k,

$$\left\|\mathbf{v}_0^k - \mathbf{x}_*\right\| \geq \epsilon \qquad \forall\, \mathbf{x}_* \in X_*\,, \qquad \forall\, k \geq 0\,.$$

Then there exists a constant $m > 0$ such that for all $k \geq 1$

$$m \leq \left\|\mathbf{v}_j^k - \mathbf{v}_l^k\right\| \qquad \forall\, 0 \leq j, l \leq n\,, \quad j \neq l\,.$$

Proof. Define the set $\mathcal{X}_*$ as follows:

$$\mathcal{X}_* = \left\{\mathbf{x} \in L(\mathbf{v}_0^0) : \|\mathbf{x} - \mathbf{x}_*\| < \frac{\epsilon}{2} \text{ for some } \mathbf{x}_* \in X_*\right\}.$$

Consider its complement Ω in $L(\mathbf{v}_0^0)$:

$$\Omega = L(\mathbf{v}_0^0) \setminus \mathcal{X}_*.$$

Note that Ω is compact. Furthermore, f is continuously differentiable on Ω, so

1. $\|\nabla f\|$ achieves a minimum $\sigma > 0$ on Ω, and
2. ∇f is uniformly continuous on Ω.

This leads to two observations. First, for all k,

$$\left\|\nabla f(\mathbf{v}_0^k)\right\| \geq \sigma > 0\,,$$

and σ does not depend on k. Second, the uniform linear independence of the set of search directions at each iteration gives us the constant $\gamma > 0$ seen in (1). From the uniform continuity of ∇f on Ω there exists a $\delta > 0$, depending only on σ, γ, and Ω, such that

$$\left\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{v}_0^k)\right\| < \sigma\gamma/2 \quad \text{whenever} \quad \left\|\mathbf{x} - \mathbf{v}_0^k\right\| < \delta \quad (\text{and } \mathbf{x} \in \Omega)\,.$$

Now we will show that once the edges of the simplex become "small enough," the rotation step is always acceptable so that the simplex will not contract; hence the simplex cannot become any smaller.

"Small enough" will mean that the simplex and its rotation are contained in Ω and $\|\mathbf{v}_j^k - \mathbf{v}_0^k\| < \delta$ for any choice of $j = 1, \cdots, n$. These conditions will determine our choice of m.

We need first some measure of the relative lengths of the edges in the simplex. We define η as follows:

$$\eta = \frac{e^k}{E^k}\,,$$

where

$$e^k = \min_{0 \leq j,l \leq n\,,\ j \neq l} \left\|\mathbf{v}_j^k - \mathbf{v}_l^k\right\|, \quad \text{and} \quad E^k = \max_{0 \leq j,l \leq n\,,\ j \neq l} \left\|\mathbf{v}_j^k - \mathbf{v}_l^k\right\|.$$

Thus, for any $1 \leq j, l \leq n$,

$$\left\|\mathbf{v}_j^k - \mathbf{v}_0^k\right\| \leq E^k = \frac{1}{\eta} e^k \leq \frac{1}{\eta} \left\|\mathbf{v}_l^k - \mathbf{v}_0^k\right\|.$$

Note that η is independent of k since, as we demonstrated in Lemma 4.2, the relative lengths of the edges in the simplex remain fixed across all iterations of the algorithm. Finally, observe that $0 < \eta \leq 1$.

We also define d to be the distance between the contour defined by $\mathbf{v}_0^0$ and the level set defined by $\mathbf{v}_0^1$:

$$d = \text{dist}\left(L(\mathbf{v}_0^1), C(\mathbf{v}_0^0)\right).$$

Note that Lemma 4.1, together with the assumption that the best vertices are uniformly bounded away from X_*, assures us that $d > 0$. Finally, we use S_k to denote the simplex defined by the vertices $\langle \mathbf{v}_0^k, \mathbf{v}_1^k, \cdots, \mathbf{v}_n^k \rangle$. We now make the following two claims.

CLAIM 1. *Suppose $k \geq 1$. If, for some $i = 1, \cdots, n$, $\|\mathbf{v}_i^k - \mathbf{v}_0^k\| \leq \eta d$, then $S_k \subset L(\mathbf{v}_0^0)$.*

Proof. The triangle inequality gives us

$$\begin{aligned}
\text{dist}\left(\mathbf{v}_j^k, L(\mathbf{v}_0^1)\right) &\leq \text{dist}\left(\mathbf{v}_j^k, L(\mathbf{v}_0^k)\right) + \text{dist}\left(L(\mathbf{v}_0^k), L(\mathbf{v}_0^1)\right) \\
&= \text{dist}\left(\mathbf{v}_j^k, L(\mathbf{v}_0^k)\right) \\
&\leq \|\mathbf{v}_j^k - \mathbf{v}_0^k\| \\
&\leq \frac{1}{\eta}\|\mathbf{v}_i^k - \mathbf{v}_0^k\| \\
&\leq d,
\end{aligned}$$

for *any* choice of $j = 1, \cdots, n$. Therefore, $S_k \in L(\mathbf{v}_0^0)$ as required. □

CLAIM 2. *If, for some $i = 1, \cdots, n$, $\|\mathbf{v}_i^k - \mathbf{v}_0^k\| \leq \eta \frac{\epsilon}{3}$, then none of the vertices in the rotated simplex are contained in $\mathcal{X}_*$, i.e., $\mathbf{r}_j^k \notin \mathcal{X}_*$ for any $j = 1, \cdots, n$.*

Proof. Recall that $\|\mathbf{r}_j^k - \mathbf{v}_0^k\| = \|\mathbf{v}_j^k - \mathbf{v}_0^k\|$ for any $j = 1, \cdots, n$.

Let $\mathbf{x}_* \in X_*$. We appeal to the "reverse" form of the triangle inequality to obtain

$$\|\mathbf{r}_j^k - \mathbf{x}_*\| \geq |\underbrace{\|\mathbf{r}_j^k - \mathbf{v}_0^k\|}_{\leq \frac{\epsilon}{3}} - \underbrace{\|\mathbf{v}_0^k - \mathbf{x}_*\|}_{\geq \epsilon}| > \frac{\epsilon}{2}.$$ □

We define a as follows:

$$a = \eta \min\left\{d, \frac{\epsilon}{3}, \delta\right\}.$$

We are now assured that if

$$\max_{j=1,\cdots,n} \|\mathbf{v}_j^k - \mathbf{v}_0^k\| \leq a, \tag{2}$$

then S_k and its rotation are contained in Ω and that $\|\nabla f(\mathbf{v}_j^k) - \nabla f(\mathbf{v}_0^k)\| < \sigma\gamma/2$ for all $j = 1, \cdots, n$. We are ready to argue that if, at any iteration k, (2) is satisfied, then the rotation step will always be acceptable. Again, we have two cases to consider.

Case 1. $\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) > 0$.

Choose a vertex $\mathbf{v}_i^k$, $i \neq 0$, which satisfies

$$\frac{\left|\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k)\right|}{\|\nabla f(\mathbf{v}_0^k)\| \, \|\mathbf{v}_i^k - \mathbf{v}_0^k\|} \geq \gamma.$$

Lemma 4.2 guarantees the existence of at least one such vertex.

By definition,

$$\mathbf{r}_i^k = \mathbf{v}_0^k - (\mathbf{v}_i^k - \mathbf{v}_0^k) \quad \text{so} \quad -(\mathbf{r}_i^k - \mathbf{v}_0^k) = \mathbf{v}_i^k - \mathbf{v}_0^k .$$

Our choice of a guarantees that the edge $(\mathbf{r}_i^k - \mathbf{v}_0^k)$ is contained in Ω. We can then invoke the Mean Value Theorem to obtain

$$f(\mathbf{r}_i^k) - f(\mathbf{v}_0^k) = \nabla f(\xi)^T (\mathbf{r}_i^k - \mathbf{v}_0^k)$$

for some $\xi \in (\mathbf{r}_i^k - \mathbf{v}_0^k)$, whence

$$f(\mathbf{r}_i^k) - f(\mathbf{v}_0^k) = \nabla f(\mathbf{v}_0^k)^T (\mathbf{r}_i^k - \mathbf{v}_0^k) + \left(\nabla f(\xi) - \nabla f(\mathbf{v}_0^k)\right)^T (\mathbf{r}_i^k - \mathbf{v}_0^k). \tag{3}$$

Consider the first term on the right-hand side of (3), $\nabla f(\mathbf{v}_0^k)^T (\mathbf{r}_i^k - \mathbf{v}_0^k)$. Our choice of $\mathbf{v}_i^k$ gives us

$$\left|\nabla f(\mathbf{v}_0^k)^T (\mathbf{v}_i^k - \mathbf{v}_0^k)\right| \geq \gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{v}_i^k - \mathbf{v}_0^k\right\| .$$

Since $\nabla f(\mathbf{v}_0^k)^T (\mathbf{v}_i^k - \mathbf{v}_0^k) > 0$ we have

$$\nabla f(\mathbf{v}_0^k)^T (\mathbf{v}_i^k - \mathbf{v}_0^k) \geq \gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{v}_i^k - \mathbf{v}_0^k\right\| .$$

By construction, $\left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| = \left\|\mathbf{v}_i^k - \mathbf{v}_0^k\right\|$. Thus,

$$\nabla f(\mathbf{v}_0^k)^T (\mathbf{v}_i^k - \mathbf{v}_0^k) \geq \gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| .$$

Since $\mathbf{v}_i^k - \mathbf{v}_0^k = -(\mathbf{r}_i^k - \mathbf{v}_0^k)$, we obtain

$$\nabla f(\mathbf{v}_0^k)^T (\mathbf{r}_i^k - \mathbf{v}_0^k) \leq -\gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| . \tag{4}$$

Now consider the second term on the right-hand side of (3). The Cauchy–Schwarz inequality gives us

$$\left|\left(\nabla f(\xi) - \nabla f(\mathbf{v}_0^k)\right)^T \left(\mathbf{r}_i^k - \mathbf{v}_0^k\right)\right| \leq \left\|\nabla f(\xi) - \nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| . \tag{5}$$

Combine (4) and (5) to rewrite (3) as

$$f(\mathbf{r}_i^k) - f(\mathbf{v}_0^k) \leq -\gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| + \left\|\nabla f(\xi) - \nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\| .$$

Since ∇f is uniformly continuous on Ω and $\|\xi - \mathbf{v}_0^k\| \leq \|\mathbf{r}_i^k - \mathbf{v}_0^k\| = \|\mathbf{v}_i^k - \mathbf{v}_0^k\| \leq a$, the following holds:

$$f(\mathbf{r}_i^k) - f(\mathbf{v}_0^k) \leq \underbrace{\underbrace{\underbrace{-\gamma \underbrace{\left\|\nabla f(\mathbf{v}_0^k)\right\|}_{\geq \sigma}}_{\leq -\sigma\gamma} + \underbrace{\left\|\nabla f(\xi) - \nabla f(\mathbf{v}_0^k)\right\|}_{< \frac{\sigma\gamma}{2}}}_{< -\frac{\sigma\gamma}{2} < 0} \underbrace{\left\|\mathbf{r}_i^k - \mathbf{v}_0^k\right\|}_{>0}}_{<0} .$$

Thus,

$$f(\mathbf{r}_i^k) - f(\mathbf{v}_0^k) < 0 \Longrightarrow f(\mathbf{r}_i^k) < f(\mathbf{v}_0^k) ,$$

whenever $\max_{j=1,\cdots,n} \|\mathbf{v}_j^k - \mathbf{v}_0^k\| \leq a$. We can then conclude that the vertex $\mathbf{r}_i^k$ can replace the best vertex $\mathbf{v}_0^k$. Therefore, the simplex will not contract.

Case 2. $\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) < 0$.

The proof for Case 2 follows that given for Case 1. The conclusion, however, is quite different. Since $\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) < 0$, we get, as in (4),

$$\nabla f(\mathbf{v}_0^k)^T(\mathbf{v}_i^k - \mathbf{v}_0^k) \leq -\gamma \left\|\nabla f(\mathbf{v}_0^k)\right\| \left\|\mathbf{v}_i^k - \mathbf{v}_0^k\right\|,$$

without substituting $-(\mathbf{r}_i^k - \mathbf{v}_0^k)$ for $\mathbf{v}_i^k - \mathbf{v}_0^k$. We continue the argument as in (5) but with $\mathbf{v}_i^k$ rather than with $\mathbf{r}_i^k$. Finishing the reductions, we are left with

$$f(\mathbf{v}_i^k) < f(\mathbf{v}_0^k)$$

whenever $\max_{j=1,\cdots,n} \|\mathbf{v}_j^k - \mathbf{v}_0^k\| \leq a$. But this leads to a contradiction! We have chosen $\mathbf{v}_i^k$ so that $i \neq 0$ and the multidirectional search algorithm requires that

$$f(\mathbf{v}_0^k) \leq f(\mathbf{v}_i^k) \qquad \forall\, i = 1, \cdots, n.$$

Therefore, Case 2 cannot happen.

Conclusion. We have now established that once every edge of the simplex has length less than a, then the rotation step will always be acceptable. Since the algorithm reduces the lengths of the edges of the simplex by at most θ at any iteration of the inner **repeat** loop, it follows that the length of every edge of the simplex will be between $\eta\theta a$ and a. Thus we choose $m = \eta\theta a$. □

We emphasize that the existence of a lower bound for the lengths of the edges in the simplex is guaranteed *only* under the null hypothesis that the best vertices are uniformly bounded away from X_*. However, now that we have established both lower and upper bounds for the lengths of the edges in the simplex, we are ready to show that, given these lower and upper bounds, the multidirectional search algorithm can only visit a finite number of points. Again, let us stress that this is not what occurs, but is rather a consequence of the hypothesis that the best vertices are uniformly bounded away from X_*.

To see how the argument works, we begin by considering how the algorithm decides which points to visit. In Fig. 1 we are given an initial simplex with a designated "best" vertex. The algorithm automatically generates the rotated simplex. The result of the acceptance test then dictates whether the expanded or the contracted simplex will be constructed. In either event, given an initial simplex and the best vertex in that simplex, we can determine, in advance, all the simplices that can be generated during the first iteration of the inner **repeat** loop. This means we can enumerate, a priori, all the points that can possibly be visited during the first iteration through the **repeat** loop.

Now assume that we are given an initial simplex, but do not know any information about the function values at any of the vertices in the simplex. This means that we have no way of identifying the "best" vertex in the simplex. Even without this information, we can still determine all the simplices that might be generated during the first iteration of the inner **repeat** loop and we can still enumerate all the points that might be visited during this iteration. To do this, we simply allow each of the vertices in the original simplex to be "best" and consider all the possibilities, as shown in Fig. 2.

We now extend this speculation further. After one pass through the inner **repeat** loop there are two possibilities: either the best vertex has been replaced, and we go to the next iteration of the outer **while** loop, or the best vertex has not been replaced and

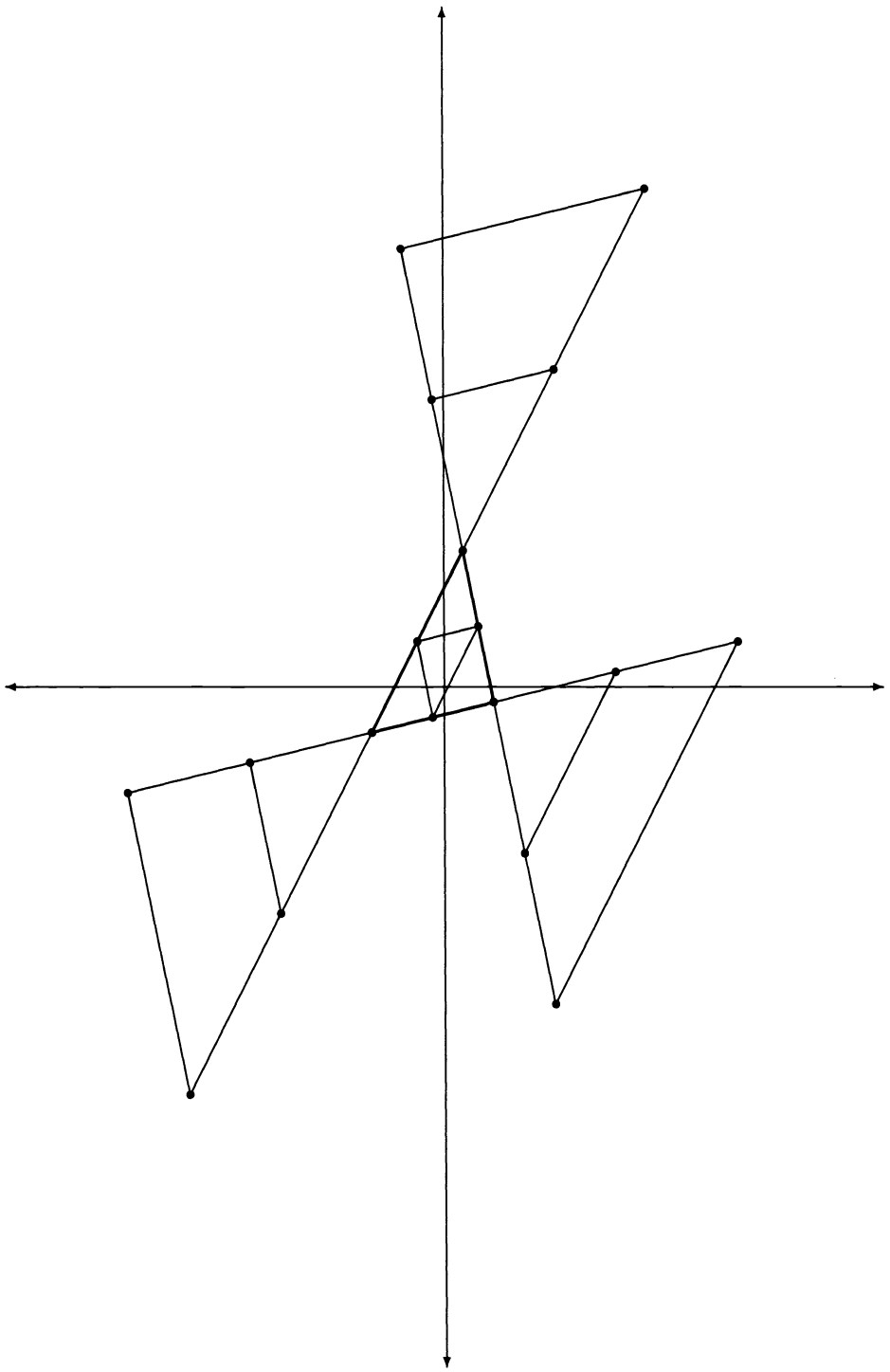


FIG. 2. *Enumerating the vertices when the best vertex is not known.*

we pass once more through the inner **repeat** loop. Again, we assume no knowledge of the function value at any of the vertices. We can still consider, a priori, all the possibilities by allowing each vertex, including the original best vertex, in each of the trial simplices generated at the previous iteration to be "best" and generate all possible new simplices. For our example, seen in Fig. 3, we begin to enforce a lower bound on the lengths of the edges in any of the simplices. Furthermore, we require our simplices to be contained in a compact set. Both of these restrictions are important because they eliminate several of the simplices that might otherwise have been considered.

Consider yet another iteration of either the outer **while** loop or the inner **repeat** loop. Again, we allow each vertex in each of the trial simplices generated at the previous iteration to be "best," but again we apply our restrictions to eliminate even more simplices. The result can be seen in Fig. 4.

Finally, if we remove all the edges, we see in Fig. 5 that the multidirectional search algorithm is, in fact, generating a grid. Since the grid must be contained in a compact set, and since its mesh size is fixed, this means that there are only a finite number of points that the algorithm can visit. We require the lower bound on the lengths of the edges in the simplex to fix the mesh size of the grid. We require the upper bound on the lengths of the edges in the simplex to give us a compact set over which to search. However, once we accept these two restrictions, this means that we can compute—without any knowledge of function information—all the points the algorithm can possibly visit from any given initial simplex.

Now we will prove that, given any initial simplex, if we assume that the best vertices are uniformly bounded away from the set X_*, then the multidirectional search algorithm can visit only a finite number of points.

PROPOSITION 5.5. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuously differentiable on $L(\mathbf{v}_0^0)$. Further, suppose that the best vertices stay uniformly bounded away from X_*, i.e., for a fixed $\epsilon > 0$, independent of k,*

$$\left\|\mathbf{v}_0^k - \mathbf{x}_*\right\| \geq \epsilon \qquad \forall\, \mathbf{x}_* \in X_*\,, \qquad \forall\, k \geq 0\,.$$

Then the multidirectional search algorithm can visit only a finite number of points.

Proof. The proof is by construction.

Rescale the initial simplex S_0, using the contraction factor θ, until every edge in the simplex satisfies the condition

$$\eta\theta a \leq \left\|\mathbf{v}_j^0 - \mathbf{v}_l^0\right\| \leq a\,,$$

for all $0 \leq j, l \leq n$, $j \neq l$, where η and a are as defined in Proposition 5.4.

Find the least common denominator for the scale factors θ and μ. This makes sense since θ and μ are restricted to the set of rational numbers.

Divide the least common denominator by the reduction factor θ. Reduce the simplex one last time by this factor.

Designate any one of the vertices in the rescaled simplex as "best." (There is no need to consider function information.) Take the set of edges adjacent to the best vertex

$$\left\{(\mathbf{v}_i^0 - \mathbf{v}_0^0) \;:\; i = 1, \cdots, n\right\}$$

as a basis for the grid. Now take all *integer* multiples of the basis that generate points inside the compact set $\mathcal{M}$. (Recall that Proposition 5.3 implies the existence of a compact set $\mathcal{M}$ which contains all the simplices generated from S_0 by the multidirectional search algorithm.)

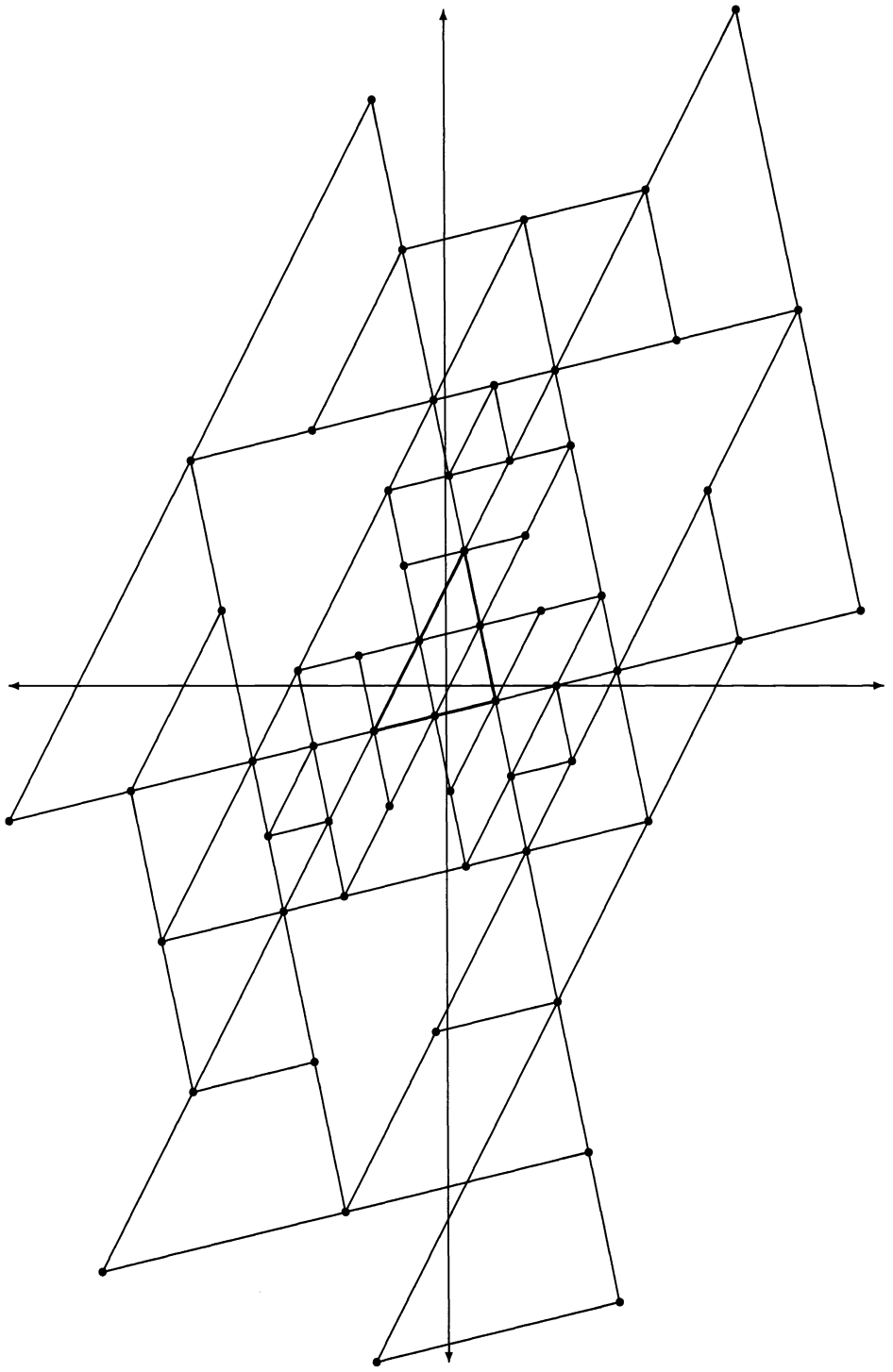


FIG. 3. *Enumerating the vertices after one additional iteration.*

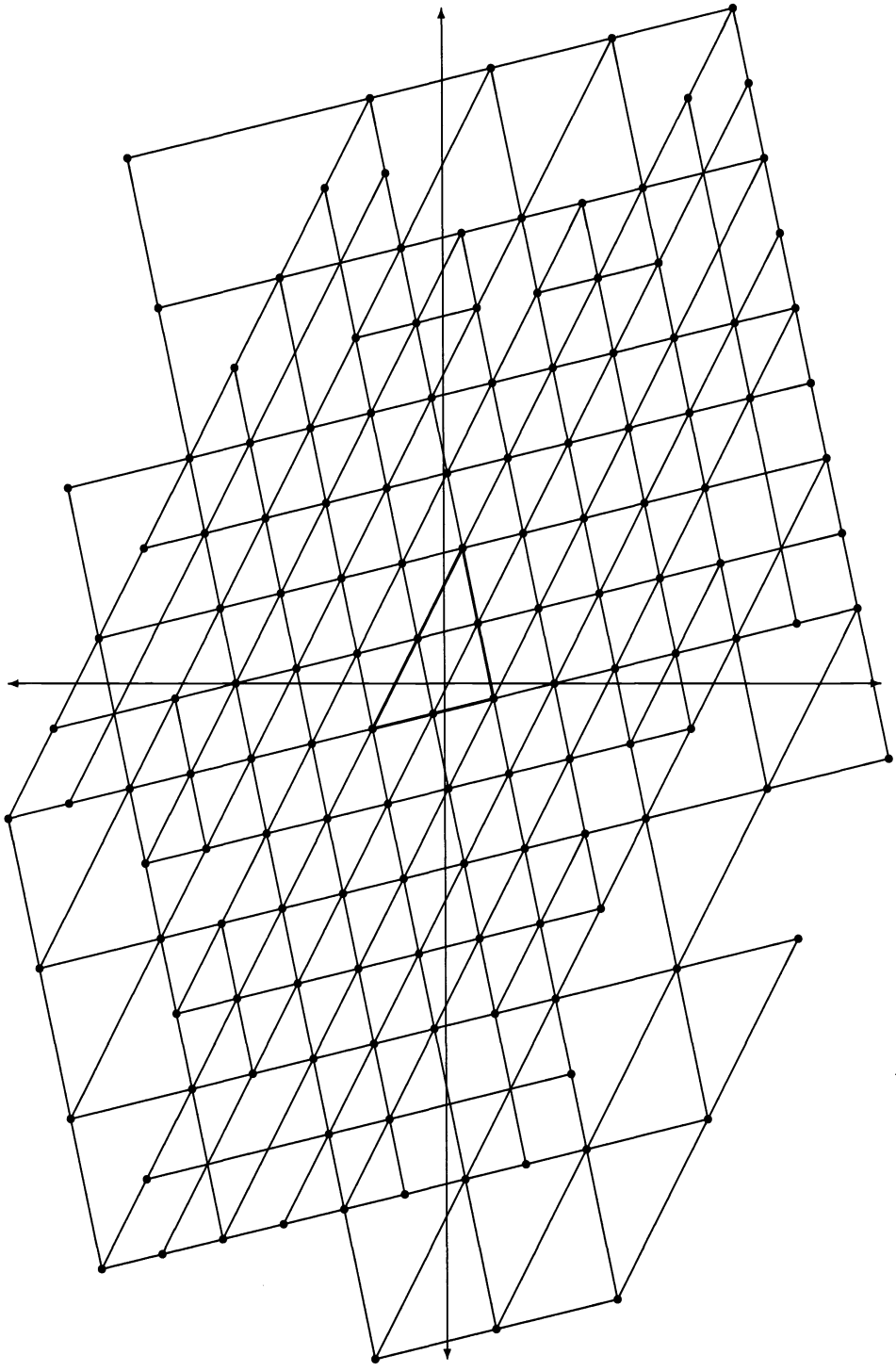


FIG. 4. *Enumerating the vertices after two additional iterations.*

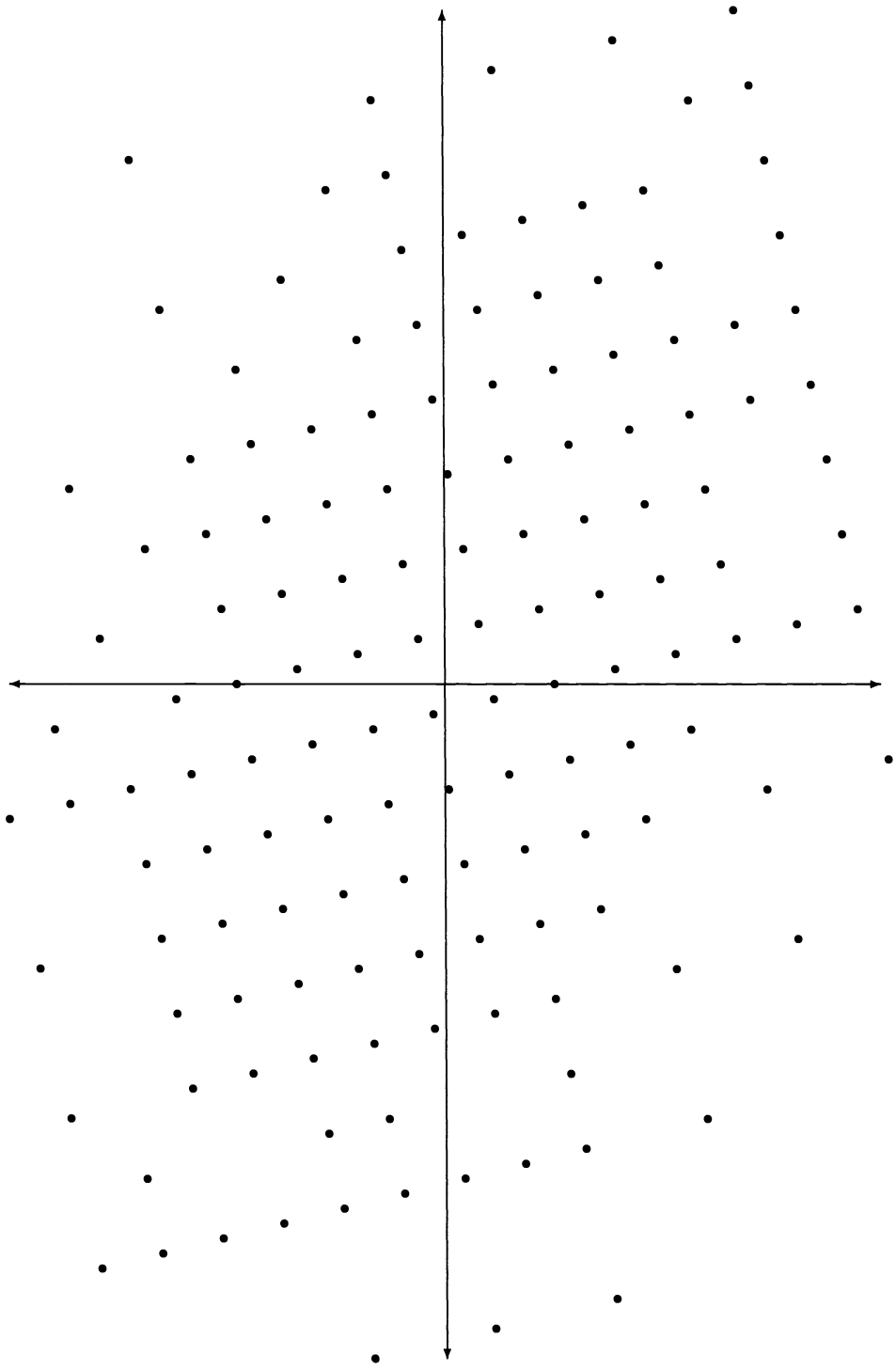


FIG. 5. *Enumerating the vertices after removing all the edges.*

This will give a grid with fixed mesh size inside a compact set. Thus the number of points in the grid will be finite. Furthermore, every possible simplex, given the initial simplex S_0, will be mapped onto the grid since all possible step sizes are integer multiples of the mesh size.

Therefore, the multidirectional search algorithm can visit only a finite number of points. □

5.2. Technical results. We now introduce one last technical result before giving the proof of Theorem 5.1.

Lemma 4.1 shows that the multidirectional search algorithm is a descent method with a strictly monotonically decreasing sequence of function values at the best vertices. The following proposition and its corollary demonstrate that this sequence of function values converges and that the sequence of best vertices converges to the corresponding level set.

PROPOSITION 5.6. *Assume that* $\{\mathbf{x}^k\}$ *is a sequence contained in a compact set* $\mathcal{S}$, *that* f *is continuous, and that* $\{f(\mathbf{x}^k)\}$ *is a monotone nonincreasing sequence. Then there is some* $\hat{f}$ *such that*

$$\lim_{k\to\infty} f(\mathbf{x}^k) = \hat{f}\,.$$

Furthermore, assume that $\hat{\mathbf{x}}$ *is any limit point of* $\{\mathbf{x}^k\}$. *Then* $f(\hat{\mathbf{x}}) = \hat{f}$ *and* $\{\mathbf{x}^k\}$ *converges to* $C(\hat{\mathbf{x}})$ *in the sense that*

$$\lim_{k\to\infty} \left[\inf_{\mathbf{y}\in C(\hat{\mathbf{x}})} \left\| \mathbf{x}^k - \mathbf{y} \right\| \right] = 0\,.$$

Proof. The proof of the first statement is immediate: $\hat{f}$ exists because $\{f(\mathbf{x}^k)\}$ is a monotone nonincreasing sequence that is bounded below.

To prove the second statement, consider the following: since $\mathcal{S}$ is compact, $\{\mathbf{x}^k\}$ contains a convergent subsequence. We will denote this convergent subsequence by $\{\mathbf{x}^{k_i}\}$ and say that $\mathbf{x}^{k_i} \to \hat{\mathbf{x}}$. By continuity, $f(\hat{\mathbf{x}}) = \lim_{i\to\infty} f(\mathbf{x}^{k_i}) = \hat{f}$.

Define

$$\delta_k = \inf_{\mathbf{y}\in C(\hat{\mathbf{x}})} \left\| \mathbf{x}^k - \mathbf{y} \right\|.$$

But $\lim_{i\to\infty} \delta_{k_i} = 0$, which implies that

$$\liminf_{k\to\infty} \delta_k = 0\,.$$

Next, suppose that $\{\delta_{k_i}\}$ is *any* convergent subsequence of $\{\delta_k\}$. There is a corresponding sequence of $\{\mathbf{x}^{k_i}\}$. This sequence has a convergent subsequence which we denote also by $\{\mathbf{x}^{k_i}\}$ and say that $\mathbf{x}^{k_i} \to \check{\mathbf{x}}$. Again, $f(\check{\mathbf{x}}) = \lim_{i\to\infty} f(\mathbf{x}^{k_i}) = \hat{f}$, so $\check{\mathbf{x}} \in C(\hat{\mathbf{x}})$. Thus, $\delta_{k_i} \to 0$. Hence, the only possible accumulation points of $\{\delta_k\}$ are 0 and ∞. Note that since $\mathcal{S}$ is compact, $\{\delta_k\}$ is bounded, i.e., there exists a $b \geq 0$ such that $0 \leq \delta_k \leq b$. Thus,

$$\limsup_{k\to\infty} \delta_k = 0\,.$$

Since $\liminf_{k\to\infty} \delta_k = \limsup_{k\to\infty} \delta_k = 0$,

$$\lim_{k\to\infty} \delta_k = 0$$

as required. □

COROLLARY 5.7. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuously differentiable on $L(\mathbf{v}_0^0)$. Then for the sequence of best vertices $\mathbf{v}_0^k$ generated by the multidirectional search algorithm there is some $\hat{f}$ such that*

$$\lim_{k\to\infty} f(\mathbf{v}_0^k) = \hat{f}\,.$$

Furthermore, assume that $\hat{\mathbf{x}}$ is any limit point of $\{\mathbf{v}_0^k\}$. Then $f(\hat{\mathbf{x}}) = \hat{f}$ and $\{\mathbf{v}_0^k\}$ converges to $C(\hat{\mathbf{x}})$ in the sense that

$$\lim_{k\to\infty} \left[\inf_{\mathbf{x}\in C(\hat{\mathbf{x}})} \left\| \mathbf{v}_0^k - \mathbf{x} \right\| \right] = 0\,.$$

Proof. Lemma 4.1 established that $\{f(\mathbf{v}_0^k)\}$ is a monotone decreasing sequence. We appeal to Proposition 5.6 to complete the proof. □

6. Proof of convergence. We have established that the multidirectional search algorithm is a descent method. In addition, we can guarantee first, that the search directions will not deteriorate, and second, that the steps taken by the algorithm cannot become too long or too short. Finally, we have shown that the sequence of function values at the best vertices converges and the sequence of best vertices converges to the corresponding level set. Thus, we are now ready to prove Theorem 5.1.

Proof. The proof is by contradiction.

Suppose that for all but finitely many k there exists a fixed constant $\epsilon > 0$, which does not depend on k, such that

$$\left\| \mathbf{v}_0^k - \mathbf{x}_* \right\| \geq \epsilon \qquad \forall\, \mathbf{x}_* \in X_*\,.$$

Then, taken together, the upper and lower bounds on the lengths of the edges in the simplex (Propositions 5.3 and 5.4) imply that the algorithm can visit only a finite number of points (Proposition 5.5). But this contradicts Lemma 4.1, which guarantees *strict* decrease in the function values at the best vertex in a finite number of iterations. Thus, the hypothesis cannot hold, which means that

$$\liminf_{k\to\infty} \left\| \mathbf{v}_0^k - \mathbf{x}_* \right\| = 0\,.$$

Then there exists some subsequence of the best vertices $\{\mathbf{v}_0^k\}$ that converges to a point $\mathbf{x}_* \in X_*$.

We invoke Corollary 5.7 to complete the proof. □

7. The nonsmooth case. To extend Theorem 5.1 to handle most cases when the function f is nondifferentiable, we need only modify the set X_*.

Let X_* include the set of stationary points of the function f in $L(\mathbf{v}_0^0)$, the set of all points in $L(\mathbf{v}_0^0)$ where f is nondifferentiable, and the set of all points in $L(\mathbf{v}_0^0)$ where the derivative of f exists but is not continuous. Our construction of the set Ω in the proof of Proposition 5.4 then guarantees that f is continuously differentiable on Ω. We can now replace the assumption that f is continuously differentiable on $L(\mathbf{v}_0^0)$ with the assumption that f is continuous on $L(\mathbf{v}_0^0)$. Since the proofs given in the previous sections were constructed to require only that ∇f be continuous on Ω, the results can be extended, without modification, to cover this new characterization of X_*.

We can state the result as follows.

THEOREM 7.1. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuous on $L(\mathbf{v}_0^0)$. Then some subsequence of $\{\mathbf{v}_0^k\}$ converges to a point $\mathbf{x}_* \in X_*$. Furthermore, $\{\mathbf{v}_0^k\}$ converges to $C_* = C(\mathbf{x}_*)$ in the sense that*

$$\lim_{k\to\infty} \left[\inf_{\mathbf{x}\in C_*} \|\mathbf{v}_0^k - \mathbf{x}\| \right] = 0 .$$

Proof. The proof follows directly from the proof of Theorem 5.1. □

This is not a general result for the nonsmooth case; rather, it is an extension of the result for the smooth case. The proof of Proposition 5.4 requires that the derivative be continuous wherever it exists to give us the uniform continuity of ∇f on the set Ω. The constant we derive from the uniform continuity of ∇f on Ω plays a key part in deriving a lower bound on the lengths of the edges in the simplex. Thus X_* must include the set of all points in $L(\mathbf{v}_0^0)$ where the derivative of f exists but is not continuous.

While this restriction is unfortunate, it does not prevent us from considering most nonsmooth cases of practical interest. For instance, it means that our theory can handle the following example constructed by Dennis and Woods [6]. Consider the strictly convex function $f(\mathbf{x}) = \frac{1}{2} \max\{\|\mathbf{x} - c_1\|^2, \|\mathbf{x} - c_2\|^2\}$, whose level sets are shown in Fig. 6, where $c_1 = (0, 32)^T$ and $c_2 = (0, -32)^T$.

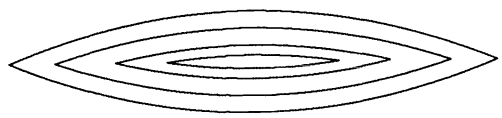


FIG. 6. *Level sets for the Dennis–Woods function.*

Our numerical experience has shown that, given any initial simplex, the multidirectional search algorithm converges unfailingly to a point of the form $(\alpha, 0)^T$ but not necessarily to the minimizer $(0, 0)^T$. Theorem 7.1 confirms that convergence to a critical point (i.e., either a point where the function is nondifferentiable or where the gradient is equal to zero) is the best that can be expected for this example.

8. Conclusions. The convergence analysis for the multidirectional search algorithm actually explains how—and why—the algorithm works. As we have seen, the fixed search directions and fixed rescaling factors used to determine the step sizes allow us to construct a grid underlying the progress of the algorithm. We should note, however, that once we remove the null hypothesis, which required the sequence of best vertices to be uniformly bounded away from the set X_*, we no longer have a lower bound on the lengths of the edges in the simplex. Thus, the algorithm allows for further and further grid refinement until a satisfactory solution is obtained.

Furthermore, once we add function information, we eliminate many of the points the algorithm might otherwise visit. This means that while there is an implicit grid structure underlying the search, the algorithm by no means visits every point on the grid. Rather, it uses function information to prune the number of grid points that are considered—and to dictate when the grid must be refined further.

The simplex that is used to start the multidirectional search algorithm determines the shape of the grid underlying the search. However, we could just as easily have used another *fixed* "shape," or "pattern," to determine the shape of the underlying grid. For example, the factorial design algorithm of Box [2], in its simplest form, constructs a hypercube centered at the current iterate and then computes the function values at the vertices in an effort to find a new iterate with a function value that is strictly less than the function value at the current iterate. If a new best point is found, the hypercube is then centered on the new best iterate and the search is restarted. If not, the size of the hypercube is reduced by halving the lengths of the edges. Again we have a pattern for the grid underlying the search. We also have all the necessary ingredients to argue for convergence of the method: fixed search directions, a fixed rescaling factor, and strict decrease in the function value at the sequence of best points.

We should be able to use the arguments presented here to analyze any algorithm that maintains this sort of underlying grid structure. This is the subject of our current research. There is, however, one additional caveat that should be added to the above discussion. The number of points required by the simplex "pattern" of the multidirectional search algorithm is $O(n)$. For the algorithms that do not use a simplex pattern, the number of points required to guarantee convergence, though not necessarily to execute the algorithm on a sequential machine, is exponential in n. (For example, the hypercube constructed in the factorial design algorithm of Box requires $O(2^n)$ new points at each iteration.) All of these algorithms share the same convergence analysis and yet the number of points required by the multidirectional search algorithm is linear, rather than exponential, in the size of the problem—which is why we believe the multidirectional search algorithm to be the most practical of these algorithms to implement on a parallel machine.

REFERENCES

[1] M. AVRIEL, *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[2] G. E. P. BOX, *Evolutionary operation: A method for increasing industrial productivity*, Appl. Statist., 6 (1957), pp. 81–101.

[3] J. CÉA, *Optimisation : théorie et algorithmes*, Dunod, Paris, 1971.

[4] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[5] J. E. DENNIS, JR. AND V. TORCZON, *Direct search methods on parallel machines*, Tech. Report 90-19, Department of Mathematical Sciences, Rice University, Houston, TX 77251–1892, 1990.

[6] J. E. DENNIS, JR. AND D. J. WOODS, *Optimization on microcomputers: The Nelder–Mead simplex algorithm*, in New Computing Environments: Microcomputers in Large-Scale Computing, A. Wouk, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987, pp. 116–122.

[7] R. HOOKE AND T. A. JEEVES, *"Direct search" solution of numerical and statistical problems*, J. Assoc. Comput. Mach., 8 (1961), pp. 212–229.

[8] J. A. NELDER AND R. MEAD, *A simplex method for function minimization*, Comput. J., 7 (1965), pp. 308–313.

[9] J. M. ORTEGA AND W. C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[10] M. J. D. POWELL, *An efficient method for finding the minimum of a function of several variables without calculating derivatives*, Comput. J., 7 (1964), pp. 155–162.

[11] H. H. ROSENBROCK, *An automatic method for finding the greatest or least value of a function*, Comput. J., 3 (1960), pp. 175–184.

[12] W. SPENDLEY, G. R. HEXT, AND F. R. HIMSWORTH, *Sequential application of simplex designs in optimisation and evolutionary operation*, Technometrics, 4 (1962), pp. 441–461.

[13] V. TORCZON, *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*, Ph.D. thesis, Department of Mathematical Sciences, Rice University, Houston, TX, 1989; also available as Tech. Report 90-7, Department of Mathematical Sciences, Rice University, Houston, TX 77251-1892.

[14] Y. WEN-CI, *The convergence property of the simplex evolutionary techniques*, Scientia Sinica, Special Issue of Mathematics, 1 (1979), pp. 68–77.

[15] ———, *Positive basis and a class of direct search techniques*, Scientia Sinica, Special Issue of Mathematics, 1 (1979), pp. 53–67.

[16] W. I. ZANGWILL, *Minimizing a function without calculating derivatives*, Comput. J., 10 (1967), pp. 293–296.

SIAM J. OPTIMIZATION
Vol. 1, No. 1, pp. 146–150, February 1991

011

A NEW PROOF OF SUPERLINEAR CONVERGENCE FOR BROYDEN'S METHOD IN HILBERT SPACE*

C.T. KELLEY† AND E.W. SACHS‡

Abstract. Broyden's method is an extension of the secant method for an equation in one real variable to an arbitrary Hilbert space setting. It is a result of Griewank that the Broyden iterates converge locally superlinearly to a root if, in addition to the assumptions needed in finite dimension, the initial approximation for the Fréchet derivative differs from the Fréchet derivative at the root only by a compact operator. In this paper a new and much simpler proof of this theorem is given based on the concept of collective compactness.

Key words. Broyden's method, superlinear convergence, collective compactness

AMS(MOS) subject classifications. 65J15, 47H17, 49D15

1. Introduction.

Broyden's method is an iterative method for the solution of finding

$$x_* \in X \text{ with } F(x_*) = 0,$$

where X is a Hilbert space with inner product $< \cdot, \cdot >$, Y is a Banach space, and $F : X \to Y$ is a Fréchet differentiable map with a Lipschitz continuous Fréchet derivative in a neighborhood of x_*. For the finite-dimensional case, in order to avoid a potentially expensive evaluation of the Jacobian of F at each iteration point, Broyden [2] considered the following iteration scheme with an update procedure for the approximating operators

$$B_n \in L(X, Y),$$

where $L(X, Y)$ denotes the space of linear bounded operators from X into Y. Given $x_n \in X, B_n \in L(X, Y)$.

(i) Solve $B_n s_n = -F(x_n)$ for $s_n \in X$.
(ii) Set $x_{n+1} = x_n + s_n \in X$ and $y_n = F(x_{n+1}) - F(x_n) \in Y$.
(iii) Set $B_{n+1} = B_n + (1/\|s_n\|^2)(y_n - B_n s_n) \otimes s_n \in L(X, Y)$.

Here, for $\bar{y} \in Y$ and $\bar{x} \in X$ the operator $\bar{y} \otimes \bar{x} \in L(X, Y)$ is defined as

$$(\bar{y} \otimes \bar{x})(x) = < \bar{x}, x > \bar{y} \text{ for all } x \in X.$$

Local convergence theorems assume that the initial data x_0 and B_0 are sufficiently close to x_* and $F'(x_*)$, respectively. Furthermore, $F'(x_*)^{-1} \in L(Y, X)$ is required, which implies that X and Y are homeomorphic to each other.

* Received by the editors October 25, 1989; accepted for publication (in revised form) September 30, 1990.

† North Carolina State University, Department of Mathematics, Box 8205, Raleigh, North Carolina 27695-8205. The research of this author was supported by National Science Foundation grants DMS-8900410 and INT-8800560, and Air Force Office of Scientific Research grant AFOSR-ISSA-890044.

‡ Universität Trier, FB IV–Mathematik, Postfach 3825, 5500 Trier, Federal Republic of Germany. The research of this author was partially supported by Air Force Office of Scientific Research grant 85-0243 and the Deutsche Forschungsgemeinschaft.

In [4] it was shown that this leads to a linear rate of convergence for the iterates. Superlinear convergence of the Broyden iterates, $\{x_n\}$,

$$\lim_{n\to\infty} \frac{\|x_{n+1} - x_*\|}{\|x_n - x_*\|} = 0, \tag{1.1}$$

for finite-dimensional problems was proved in [3]. However, Stoer [10] proved with counterexamples that quasi-Newton methods do not yield superlinear convergence in Hilbert space under the same assumptions as in the finite-dimensional case. It was shown in [7] that the Dennis–Moré condition is still necessary and sufficient for superlinear convergence in Hilbert space. But this condition could only be verified for Broyden's method in a weak sense, which is identical with the strong version if X is a finite-dimensional space. In [11] a proof of superlinear convergence was given for the linear case where $D_0 = B_0 - F'(x_*)$ has p-summable singular values. In [9] it was proved that the superlinear rate of convergence is retained under the additional assumption that D_0 is a Hilbert–Schmidt operator. Griewank ([5],[6]) weakened this assumption to D_0 being a compact operator. He proved this result using the singular values and their variational characterization. His analysis could be applied to variable metric rank-two updates as well as Broyden's method. Here we give a different proof for the Broyden iterates which does not require singular value information and is more elementary. Our proof is based on the fact that the sequence of discrepancies $D_n = B_n - F'(x_*)$ is a collectively compact set of operators if D_0 is compact. This proof is also valid for general Hilbert spaces without the separability assumption that was needed in the proof in [6].

2. Convergence rate results. First, we state the linear convergence rate theorem due to Dennis [4].

THEOREM 2.1. *Let F be Fréchet differentiable and $F'(\cdot)$ Lipschitz continuous in a neighborhood of x_* with $F(x_*) = 0$ and $F'(x_*)^{-1} \in L(Y, X)$.*

Then for each $c \in (0,1)$ there exists $\varepsilon > 0$ such that if

$$\|x_0 - x_*\| \le \varepsilon \quad \text{and} \quad \|B_0 - F'(x_*)\| \le \varepsilon,$$

then the Broyden iterates are well defined and satisfy

$$\|x_{n+1} - x_*\| \le c\|x_n - x_*\| \tag{2.1}$$

for all $n \in \mathbb{N}$.

The next theorem gives a condition for a superlinear rate of convergence; see also [7].

THEOREM 2.2. *Let all the assumptions of Theorem 2.1 be satisfied and assume that the Broyden iterates satisfy (2.1) with $c \in (0,1)$. Then we obtain a superlinear rate of convergence, (1.1), if and only if the Dennis–Moré condition*

$$\lim_{n\to\infty} \frac{\|(B_n - F'(x_*))s_n\|}{\|s_n\|} = 0 \tag{2.2}$$

holds.

In general, however, for Broyden's method the last condition is only true in the sense of weak convergence; see also [7]. Thus, we have Theorem 2.3.

THEOREM 2.3. *Under the assumptions of Theorem 2.2, the iterates produced by Broyden's method satisfy*

$$\lim_{n\to\infty} \frac{\lambda((B_n - F'(x_*))s_n)}{\|s_n\|} = 0$$

for all $\lambda \in Y'$, the dual of Y.

In the next lemma we show that $D_n = B_n - F'(x_*)$ is a sequence of collectively compact operators (see also [1]), provided D_0 is compact. Recall that a sequence of linear operators, $\{K_n\}$, is collectively compact if for every bounded set, $\mathcal{B}$, the set, $\cup_n K_n \mathcal{B}$, is relatively compact. Note that since any bounded set is a subset of a scalar multiple of $\mathcal{B}(1)$ (the closed ball of radius one about zero in X), the collective compactness of $\{K_n\}$ is equivalent to the relative compactness of the set

$$\cup_n K_n \mathcal{B}(1).$$

LEMMA 2.4. *Let all the assumptions of Theorem 2.2 be satisfied and let D_0 be compact. Then the sequence $\{D_n\}$ is collectively compact.*

Proof. As above, we let $\mathcal{B}(\rho)$ denote the closed ball of radius ρ about 0 in X. To prove the assertion we exhibit a compact set S such that

$$\cup_n S_n \subset S, \quad \text{where} \quad S_n = D_n \mathcal{B}(1).$$

Let $P_n = s_n \otimes s_n / \|s_n\|^2 \in L(X, X)$. Note that the update formula for B_n yields

$$D_{n+1} = D_n(I - P_n) + \Delta_n,$$

where Δ_n is the rank-one operator

$$\Delta_n = \frac{\int_0^1 (F'(x_n + ts_n) - F'(x^*))s_n \, dt \otimes s_n}{\|s_n\|^2}.$$

Now, $D_n(I - P_n)\mathcal{B}(1) \subset D_n \mathcal{B}(1)$ because P_n is an orthogonal projector on X, so that

$$S_{n+1} \subset S_n + \Delta_n \mathcal{B}(1).$$

Moreover, by linear convergence (2.1) of the Broyden iterates, $\|\Delta_n\| \leq Kc^n$ for some $K > 0$ and $c \in (0, 1)$. Hence for each n there is an $\eta_n \in X$ with $\|\eta_n\| = 1$ such that if we let

$$I_n = \{x \in X \mid x = \alpha \eta_n, \ -1 \leq \alpha \leq 1\},$$

then

$$S_{n+1} \subset S_n + Kc^n I_n.$$

Therefore for all n,

$$S_n \subset S_0 + \sum_{j=0}^{\infty} Kc^j I_j = S.$$

The set $\tilde{S} = \sum_{j=0}^{\infty} c^j I_j$ is compact because $\sum_{j=0}^{\infty} c^j$ converges. Granting this point, the compactness of S is shown because S_0 is compact as D_0 is a compact operator.

To complete the proof we verify that $\tilde{S} \sum_{j=0}^{\infty} c^j I_j$ is compact. The proof is by a standard diagonalization argument (see [8] for other uses of this technique in functional analysis) and is put in only for completeness. Let $\{x_l\} \subset \tilde{S}$. We will construct a convergent subsequence of $\{x_l\}$. We may represent each element of the sequence by

$$x_l = \sum_{j=0}^{\infty} c^j \alpha_{l,j} \eta_j.$$

where $\{\alpha_{l,j}\} \subset [-1,1]$. As $[-1,1]$ is compact there is, for each j, a subset of the natural numbers, $\mathcal{N}_j$, such that a subsequence of $\{\alpha_{l,j}\}_{l\in\mathcal{N}_j}$ is convergent to a limit, α_j^*. Clearly we may form these sequences in such a way that $\mathcal{N}_j \subset \mathcal{N}_{j-1}$ by extracting the convergent subsequence of $\{\alpha_{l,j-1}\}$ before that of $\{\alpha_{l,j}\}$. Let

$$\tilde{x} = \sum_{j=0}^{\infty} c^j \alpha_j^* \eta_j$$

and

$$\tilde{x}_k = \sum_{j=0}^{\infty} c^j \alpha_{l(k),j} \eta_j$$

where $l(k)$ is the kth element of $\mathcal{N}_k$. By the nesting of the subsequences, $\mathcal{N}_j \subset \mathcal{N}_{j-1}$, $\{\tilde{x}_k\}$ is a subsequence of $\{x_l\}$. We complete the proof by showing that $\tilde{x}_k \to \tilde{x}$.

Let $\epsilon > 0$ be given. First let K_0 be such that

$$\sum_{j=K_0+1}^{\infty} c^j = c^{K_0+1}/(1-c) < \epsilon/4.$$

Then for $k > K_0$

$$\|\tilde{x}_k - \tilde{x}\| \le \sum_{j=0}^{K_0} |\alpha_{l(k),j} - \alpha_j^*| + 2 \sum_{j=K_0+1}^{\infty} c^j \le \sum_{j=0}^{K_0} |\alpha_{l(k),j} - \alpha_j^*| + \epsilon/2.$$

Now let K_1 be such that

$$|\alpha_{l(k),j} - \alpha_j^*| < \epsilon/(K_0+1)$$

for all $k > K_1$ and $j \le K_0$. Then if $k > \max(K_0, K_1)$,

$$\|\tilde{x}_k - \tilde{x}\| < \epsilon. \qquad \square$$

THEOREM 2.5. *If the assumptions of Lemma* 2.4 *hold then the Broyden iterates converge q-superlinearly to* x_*.

Proof. Lemma 2.4 implies that every subsequence of

$$q_n = D_n(s_n/\|s_n\|) \in S_n \subset S$$

has a norm convergent subsequence. Since q_n converges weakly to zero by Theorem 2.3, this observation implies that the whole sequence q_n converges to zero in norm and hence that the Dennis–Moré condition (2.2) holds. Theorem 2.2 yields the statement of the theorem. $\square$

REFERENCES

[1] P. M. ANSELONE, *Collectively Compact Operator Approximation Theory*, Prentice-Hall, Englewood Cliffs, NJ, 1971.

[2] C. G. BROYDEN, *A class of methods for solving simultaneous equations*, Math. Comp., 19 (1965), pp. 577–593.

[3] C. G. BROYDEN, J. E. DENNIS, AND J. J. MORÉ, *On the local and superlinear convergence of quasi-Newton methods*, IMA J., 12 (1973), pp. 223–246.

[4] J. E. DENNIS, *Toward a unified convergence theory for Newton-like methods*, in Nonlinear Functional Analysis and Applications, L. B. Rall, ed., Academic Press, New York, 1971, pp. 425–472.

[5] A. GRIEWANK, *Rates of convergence for secant methods on nonlinear problems in Hilbert space*, Proc. Numerical Analysis, Guanajuato, Mexico, 1984, in Lecture Notes in Mathematics 1230, J. P. Hennart, ed., Springer-Verlag, Heidelberg, 1986, pp. 138–157.

[6] ———, *The local convergence of Broyden-like methods on Lipschitzian problems in Hilbert space*, SIAM J. Numer. Anal., 24 (1987), pp. 684–705.

[7] W. A. GRUVER AND E. SACHS, *Algorithmic Methods in Optimal Control*, Pitman, London, 1980.

[8] M. REED AND B. SIMON, *Methods of Modern Mathematical Physics, I: Functional Analysis*, Academic Press, London, 1972.

[9] E. SACHS, *Broyden's method in Hilbert space*, Math. Prog., 35 (1986), pp. 71–82.

[10] J. STOER, *Two examples on the convergence of certain rank-2 minimization methods for quadratic functionals in Hilbert space*, Linear Algebra Appl., 28 (1979), pp. 217–222.

[11] R. WINTHER, *A numerical Galerkin method for a parabolic problem*, Ph.D. thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1977.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 151-165, May 1991

A SIMPLICIAL ALGORITHM FOR THE NONLINEAR STATIONARY POINT PROBLEM ON AN UNBOUNDED POLYHEDRON*

Y. DAI†, G. VAN DER LAAN‡, A. J. J. TALMAN§, AND Y. YAMAMOTO†¶

Abstract. A path-following algorithm is proposed for finding a solution to the nonlinear stationary point problem on an unbounded, convex, and pointed polyhedron. The algorithm can start at an arbitrary point of the polyhedron. The path to be followed by the algorithm is described as the path of zeros of some piecewise continuously differentiable function defined on an appropriate subdivided manifold. This manifold is induced by a generalized primal-dual pair of subdivided manifolds. The path of zeros can be approximately followed by dividing the polyhedron into simplices and replacing the original function by its piecewise linear approximation with respect to this subdivision. The piecewise linear path of this function can be generated by alternating replacement steps and linear programming pivot steps. A condition under which the path of zeros converges to a solution is also stated, and a description of how the algorithm operates when the problem is linear or when the polyhedron is the Cartesian product of a polytope and an unbounded polyhedron is given.

Key words. simplicial algorithm, stationary point problem, subdivided manifold, convergence condition

AMS(MOS) subject classifications. 90A14, 90A30, 90A33

1. Introduction. Let K be a convex polyhedron in R^n. We assume that K is unbounded and pointed, i.e., K has at least one vertex, and that K is represented by the set $\{x \in R^n \mid A^t x \leqq b\} = \{x \in R^n \mid (a_i)^t x \leqq b_i \text{ for } i = 1, \cdots, m\}$, where A is an $n \times m$ matrix consisting of column vectors a_i for $i = 1, \cdots, m$, and $b = (b_1, \cdots, b_m)^t$ is an m-vector. Further, let f be a continuously differentiable function from K to R^n. Then the (nonlinear) *stationary point problem* for f on K is to find a point x in K such that

$$(z-x)^t f(x) \geqq 0$$

for any point z in K. We call x a *stationary point* of f on K. If the function f is affine on K, we call the problem the *linear stationary point problem.* The stationary point problem on an unbounded convex polyhedron is frequently met in mathematical programming, for example, to find a Karush-Kuhn-Tucker point for an optimization problem with linear constraints.

To solve the nonlinear stationary point problem on K we propose a path-following algorithm. Such an algorithm traces the set of zeros of a piecewise continuously differentiable function g defined from an $(n+1)$-dimensional subdivided manifold to R^n. In case the zero vector is a regular value of the function g there exists a path of zeros initiating from an arbitrarily chosen point in K. The $(n+1)$-dimensional subdivided manifold is induced by a generalized primal-dual pair of subdivided manifolds, where the primal sets are determined by the faces of K and the dual sets are determined by the normal cones of these faces. A primal-dual pair of subdivided manifolds is a basic framework in path-following techniques for finding fixed points or solving stationary point problems (see, for example, [1], [2], [4], [7]-[9], and [10]).

* Received by the editors October 30, 1989; accepted for publication (in revised form) August 16, 1990.
† Institute of Socio-Economic Planning, University of Tsukuba, Tsukuba, Ibaraki 305, Japan.
‡ Department of Econometrics, Vrije Universiteit, P.O. Box 7161, 1007 MC Amsterdam, the Netherlands.
§ Department of Econometrics, Tilburg University, P.O. Box 90153, 5000 LE Tilburg, the Netherlands.
¶ The research of this author was partly supported by Alexander von Humboldt-Stiftung while he visited Forschungsinstitut für Diskrete Mathematik/Institut für Ökonometrie und Operations Research, Universität Bonn, Federal Republic of Germany.

The path S of zeros of the function can be approximately followed by a simplicial algorithm. This algorithm subdivides first the set K into simplices in some appropriate way and replaces the function f by its piecewise linear approximation $\bar{f}$ with respect to this triangulation. For this function the path of zeros of g becomes piecewise linear and can therefore be followed by making alternating replacement steps and linear programming pivot steps for a sequence of adjacent simplices of varying dimension.

Since the set K is unbounded, the path S may diverge to infinity. We state a simple condition on the function under which the path S is bounded and therefore leads from the starting point to a solution of the problem. We also describe how the algorithm should be adapted in case K is the Cartesian product of a polytope, i.e., a bounded convex polyhedron, and an unbounded convex polyhedron, and under what condition the path S is bounded for this case. We conclude the paper with a short description of the algorithm when the function is affine on K. The convergence condition for this problem is related to the well-known condition of copositive plus in case of the linear complementarity problem.

This paper is a generalization of path-following techniques introduced earlier for solving stationary point problems. In [8] such a method has been proposed for the linear stationary point problem on a polytope. In [7] the nonlinear stationary point problem on a polytope was treated. Finally, in [1] a path-following algorithm for the linear stationary point problem on a polyhedral cone was introduced.

This paper is organized as follows. Section 2 briefly reviews a basic theorem for path-following algorithms and extends the concept of a primal-dual pair of subdivided manifolds. In § 3 we describe the generalized pair of primal-dual subdivided manifolds which will underlie the algorithm. Section 4 defines the path of zeros leading from an arbitrary point to either infinity or a solution. We describe how this path can be approximately followed by a simplicial algorithm. In § 5 we state a convergence condition guaranteeing that the path is bounded. Finally, §§ 6 and 7 discuss the cases when K is the product of a polytope and a convex polyhedron and when f is affine on K, respectively.

2. Generalization of the primal-dual pair of subdivided manifolds. We shall briefly review a basic theorem for path-following algorithms and extend the concept of the primal-dual pair of subdivided manifolds introduced by Kojima and Yamamoto [4].

We call an l-dimensional convex polyhedron a *cell* or an *l-cell.* When a cell X is a face (see, for example, [6]) of a cell Y, we write $X \leq Y$. We denote $X < Y$ when X is a proper face of Y. Particularly when an $(l-1)$-cell X is a face of an l-cell Y, we call X a *facet* of Y and denote it by $X \lhd Y$.

A collection $\mathcal{L}$ of cells of the same dimension, say l, is called an l-dimensional subdivided manifold if it satisfies the following conditions:

(1) any two cells of $\mathcal{L}$ intersect in a common face unless the intersection is empty,

(2) any facet of a cell of $\mathcal{L}$ lies in at most two cells of $\mathcal{L}$,

(3) each point of cells of $\mathcal{L}$ has a neighborhood which intersects finitely many cells of $\mathcal{L}$.

We denote the collection of all faces of cells of $\mathcal{L}$ by $\bar{\mathcal{L}}$, i.e.,

$$\bar{\mathcal{L}} = \{X \mid X \text{ is a face of some cell of } \mathcal{L}\},$$

and the union of all cells of $\mathcal{L}$ by $|\mathcal{L}|$, i.e.,

$$|\mathcal{L}| = \bigcup [X \mid X \text{ is a cell of } \mathcal{L}].$$

It is noteworthy that $\bar{\mathcal{L}}$ consists of cells of various dimensions. By the second and most crucial condition, each $(l-1)$-cell of $\bar{\mathcal{L}}$ lies in either one or two l-cells of $\mathcal{L}$. We

refer to the collection of those $(l-1)$-cells lying in exactly one l-cell of $\mathscr{L}$ as the *boundary* of $\mathscr{L}$ and denote it by $\partial\mathscr{L}$. A continuous mapping g from $|\mathscr{L}|$ into R^n is *piecewise continuously differentiable* (abbreviated by PC^1) on $\mathscr{L}$ if the restriction of g to each cell of $\mathscr{L}$ has a continuously differentiable extension. We denote the Jacobian matrix of g at a point x of a cell C of $\mathscr{L}$ by $Dg(x; C)$. A point $c \in R^n$ is a *regular value* of the PC^1 mapping $g: |\mathscr{L}| \to R^n$ if

$$x \in B \leq C \in \mathscr{L} \quad \text{and} \quad g(x) = c \quad \text{imply} \quad \dim\{Dg(x; C)y \mid y \in B\} = n.$$

We now state one of the basic theorems for a path-following algorithm [2].

THEOREM 2.1. *Let $\mathscr{L}$ be an $(n+1)$-dimensional subdivided manifold in some Euclidean space and let $g: |\mathscr{L}| \to R^n$ be a PC^1 mapping. Suppose $c \in R^n$ is a regular value of g and $g^{-1}(c) \neq \emptyset$. Then $g^{-1}(c)$ is a disjoint union of paths and loops, where a path is a connected one-dimensional manifold homeomorphic to one of the intervals $(0, 1)$, $(0, 1]$, and $[0, 1]$, and a loop is a connected one-dimensional manifold homeomorphic to the one-dimensional sphere. Furthermore, $g^{-1}(c)$ has the following properties*:

(1) *$g^{-1}(c) \cap X$ is either empty or a disjoint union of smooth one-manifolds for each $X \in \mathscr{L}$,*

(2) *a loop of $g^{-1}(c)$ does not intersect $|\partial\mathscr{L}|$,*

(3) *if a path S of $g^{-1}(c)$ is compact, the boundary ∂S of S consists of two distinct points in $|\partial\mathscr{L}|$.*

We first generalize the primal-dual pair of subdivided manifolds proposed in [4]. In [4] the dual operator relating a pair of subdivided manifolds was assumed to satisfy several conditions including one-to-one. We will here relax these conditions. Let $\mathscr{P}$ and $\mathscr{D}$ be subdivided manifolds. A dual operator, say d, is defined on $\bar{\mathscr{P}}$ and assigns to each cell of $\bar{\mathscr{P}}$ either the empty set or a cell Y of $\bar{\mathscr{D}}$ such that for some fixed positive integer l, called the degree,

$$\dim X + \dim Y = l$$

holds. We denote the image of $X \in \bar{\mathscr{P}}$ under the operator d by X^d. When a pair of subdivided manifolds $\mathscr{P}$ and $\mathscr{D}$ is linked by such an operator d, we call the triplet $(\mathscr{P}, \mathscr{D}; d)$ a *generalized primal-dual pair of subdivided manifolds*, GPDM for short. We allow a dual operator to assign the same cell of $\bar{\mathscr{D}}$ to more than one cell of $\bar{\mathscr{P}}$, that is, to be a noninjective dual operator. Letting

$$\mathscr{L} = \{X \times X^d \mid X \in \bar{\mathscr{P}}, X^d \neq \emptyset\}, \tag{2.1}$$

the conditions required for $\mathscr{L}$ to be a subdivided manifold are given in the next lemma.

LEMMA 2.2. *Suppose $(\mathscr{P}, \mathscr{D}; d)$ is a* GPDM *with degree l. Let $\mathscr{L}$ be defined by* (2.1). *Then $\mathscr{L}$ is an l-dimensional subdivided manifold if and only if for every $(l-1)$-cell $X \times Y$ of $\bar{\mathscr{L}}$*:

(1) *there are at most two cells Z of $\bar{\mathscr{P}}$ such that*

$$X \lhd Z \quad \text{and} \quad Z^d = Y, \tag{2.2}$$

(2) *if $Y \lhd X^d$, then there is at most one cell Z of $\bar{\mathscr{P}}$ satisfying* (2.2).

Proof. Among the three conditions of a subdivided manifold the second one is crucial and the others will be straightforward. Note that an $(l-1)$-cell $X \times Y$ of $\bar{\mathscr{L}}$ is a facet of an l-cell $Z \times Z^d$ of $\mathscr{L}$ if and only if either

$$X = Z \quad \text{and} \quad Y \lhd Z^d \tag{2.3}$$

or

$$X \lhd Z \quad \text{and} \quad Y = Z^d \tag{2.4}$$

holds. From the first condition (1) it follows that there are at most two cells $Z \times Z^d$ of $\mathscr{L}$ satisfying (2.4). Furthermore, condition (2) means that there is at most one such cell if a cell $Z \times Z^d = X \times X^d$ satisfying (2.3) exists. Therefore we have shown that $X \times Y$ lies in at most two l-cells of $\mathscr{L}$.

The "only if" part is readily seen by the same argument. □

The following lemma characterizes the cells constituting the boundary $\partial\mathscr{L}$ of $\mathscr{L}$.

LEMMA 2.3. *An $(l-1)$-cell $X \times Y$ of $\bar{\mathscr{L}}$ belongs to the boundary $\partial\mathscr{L}$ of $\mathscr{L}$ if and only if the following conditions hold*:

(1) *if $Y \lhd X^d$, then there is no cell Z of $\bar{\mathscr{P}}$ satisfying* (2.2),

(2) *if $Y \not\lhd X^d$, then there is exactly one cell Z of $\bar{\mathscr{P}}$ satisfying* (2.2).

3. Construction of a GPDM. In what follows, we shall present a subdivision of the polyhedron K and construct a GPDM having this subdivision as the primal subdivided manifold. We assume that K is unbounded.

It is well known that K can be decomposed into a polytope and a polyhedral cone C containing the directions of all rays in K, and that C is given by $C = \{x \mid A^t x \leqq 0\}$ (see, for example, [6]). Since K is pointed, the cone C of rays is also pointed, namely, $C \cap (-C) = \{0\}$. Indeed, suppose that $r \in C \cap (-C)$ and consider two points $v + r$ and $v - r$ for an arbitrarily chosen vertex v of K. Since r and $-r \in C$, both of these two points lie in K. If $r \neq 0$, then the point v would be a middle point of these points, which contradicts the fact that v is a vertex.

Let w be an arbitrary point of K. In the algorithm proposed below for solving the stationary point problem on K the point w will be the starting point. For some strictly positive m-vector γ, let $h = -A\gamma$ and let $H^0 = \{x \mid h^t x = h_0\}$ be a hyperplane for some positive number h_0. We can see that if h_0 is sufficiently large, this hyperplane intersects every unbounded face of K while the negative halfspace $H^- = \{x \mid h^t x \leqq h_0\}$ contains w and all vertices of K in its interior and hence all bounded faces of K. To see this, let r be a nonzero vector of C. Since C is pointed, $A^t r \neq 0$. More precisely, $A^t r \leqq 0$ and $(a_i)^t r < 0$ for at least one column a_i of A. Then, by the definition of h,

$$h^t r = -\gamma^t A^t r > 0. \tag{3.1}$$

Therefore, when h_0 is large enough for the interior of H^- to contain all vertices of K and w, the hyperplane H^0 intersects every unbounded face of K.

Now we introduce several notations. Let $H^+ = \{x \mid h^t x \geqq h_0\}$ be the positive halfspace of H^0. For any face F of K, let

$$F^- = \{x \mid x \in F \cap H^-\},$$

$$F^0 = \{x \mid x \in F \cap H^0\},$$

and

$$F^+ = \{x \mid x \in F \cap H^+\}.$$

Note that if some face F is entirely included in H^-, then $F^- = F$. For an arbitrary subset G of K, we denote the convex hull of G and w by wG. Let

$$\mathscr{P} = \{wF^- \mid w \notin F \lhd K\} \cup \{wK^0\} \cup \{K^+\}. \tag{3.2}$$

It can easily be proved that $\mathscr{P}$ is a subdivided manifold with the same dimension as K. Moreover, the collection $\bar{\mathscr{P}}$ is equal to

$$\begin{aligned}\bar{\mathscr{P}} = &\{wF^- \mid w \notin F < K\}\\ &\cup \{wF^0 \mid F \text{ is an unbounded face of } K\}\\ &\cup \{F^+ \mid F \text{ is an unbounded face of } K\}\\ &\cup \{F^0 \mid F \text{ is an unbounded face of } K\}\\ &\cup \{F^- \mid w \notin F < K\}\\ &\cup \{w\},\end{aligned} \tag{3.3}$$

and

$$|\mathscr{P}| = K. \tag{3.4}$$

An example is illustrated in Fig. 3.1.

To make the dual subdivided manifold $\mathscr{D}$, we subdivide R^n in almost the same way as in [1]. The *normal cone* at $x \in K$ to K is defined to be

$$N(x, K) = \{y \mid y^t(z - x) \leqq 0 \text{ for every } z \in K\}. \tag{3.5}$$

It is the cone of all outward normal vectors at x to K. It is readily seen that normal cones are identical at any relative interior point of a face F of K. Therefore we denote it by F^*. Letting

$$I(F) = \{i \mid (a_i)^t x = b_i \text{ for every } x \in F\},$$

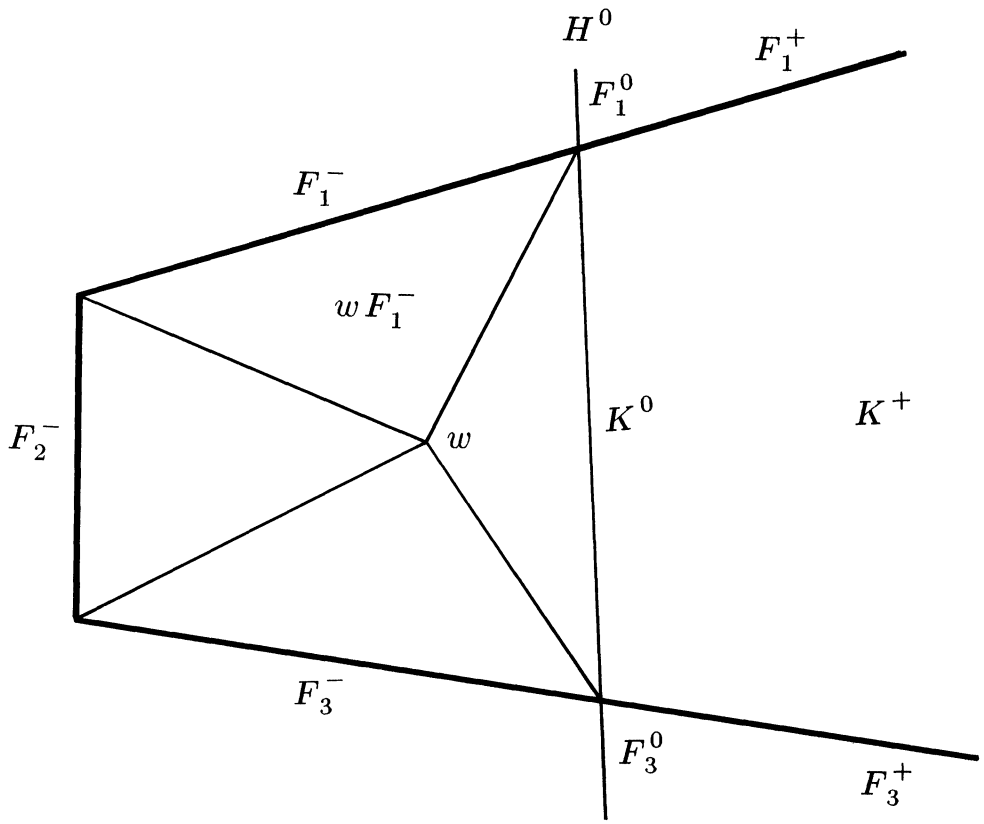


FIG. 3.1

then we obtain

$$F^* = \left\{ y \mid y = \sum_{i \in I(F)} \mu_i a_i,\ \mu_i \geqq 0 \text{ for each } i \in I(F) \right\}.$$

The dual subdivided manifold $\mathscr{D}$ is defined to be

$$\begin{aligned} \mathscr{D} = &\{\{v\}^* \mid v \text{ is a vertex of } K\} \\ &\cup \{F^* + \langle h \rangle \mid F \text{ is an extreme ray of } K\}, \end{aligned} \tag{3.6}$$

where

$$\langle h \rangle = \{y \mid y = \alpha h \text{ for some } \alpha \geqq 0\}, \tag{3.7}$$

being the ray in the direction h. Then $\mathscr{D}$ is obviously an n-dimensional subdivided manifold,

$$\begin{aligned} \bar{\mathscr{D}} = &\{F^* \mid F \leq K\} \\ &\cup \{F^* + \langle h \rangle \mid F \text{ is an unbounded face of } K\}, \end{aligned} \tag{3.8}$$

and

$$|\mathscr{D}| = R^n. \tag{3.9}$$

For constructing a GPDM it remains to define an operator d linking the subdivided manifolds $\mathscr{P}$ and $\mathscr{D}$. Let

$$\begin{aligned} &(wF^-)^d = F^* && \text{if } w \notin F < K; \\ &(wF^0)^d = F^* + \langle h \rangle && \text{if } F \text{ is an unbounded face of } K; \\ &(F^+)^d = F^* + \langle h \rangle && \text{if } F \text{ is an unbounded face of } K; \\ &(F^-)^d = \varnothing && \text{if } w \notin F < K; \\ &(F^0)^d = \varnothing && \text{if } F \text{ is an unbounded face of } K; \\ &(\{w\})^d = \varnothing. \end{aligned} \tag{3.10}$$

Then the dimensions of a cell X in $\bar{\mathscr{P}}$ and its dual cell X^d in $\bar{\mathscr{D}}$ sum up to $n+1$ if X^d is nonempty, that is, the GPDM $(\mathscr{P}, \mathscr{D}; d)$ constructed above has degree $n+1$. Let $\mathscr{M}$ be the collection of $(n+1)$-dimensional cells defined by (2.1) for this GPDM $(\mathscr{P}, \mathscr{D}; d)$. We shall show that $\mathscr{M}$ is an $(n+1)$-dimensional subdivided manifold by demonstrating that the GPDM $(\mathscr{P}, \mathscr{D}; d)$ satisfies the conditions of Lemma 2.2.

LEMMA 3.1. *For any n-cell $X \times Y$ of $\bar{\mathscr{M}}$ derived from* (3.3), (3.8), *and* (3.10), *the two conditions* (1) *and* (2) *of Lemma* 2.2 *are satisfied.*

Proof. From the definition of the dual operator d it follows that if at least two cells of $\bar{\mathscr{P}}$ are mapped to an identical cell they must be equal to wF^0 and F^+ for some unbounded face F of K. This means that condition (1) of Lemma 2.2 is satisfied. Next, suppose that there are two different cells Z_1 and Z_2 in $\bar{\mathscr{P}}$ satisfying (2.2). Then, $Z_1 = wF^0$ and $Z_2 = F^+$ for some unbounded face F of K. Since X is a facet of both Z_1 and Z_2, X must be F^0 and hence $X^d = \varnothing$. This proves that the second condition of Lemma 2.2 is also satisfied. □

Thus we have seen that $\mathcal{M}$ is an $(n+1)$-dimensional subdivided manifold as an immediate consequence of Lemma 2.2. By applying Lemma 2.3 to the GPDM $(\mathcal{P}, \mathcal{D}; d)$ considered here, we obtain the following lemma.

LEMMA 3.2.

$$
\begin{aligned}
\partial\mathcal{M} = &\{\{w\}\times F^* \mid w \notin F \prec K,\ \dim F = 0\} \\
&\cup \{\{w\}\times (F^* + \langle h\rangle) \mid F \textit{ is an extreme ray of } K\} \\
&\cup \{F^+ \times F^* \mid F \textit{ is an unbounded face of } K\} \\
&\cup \{F^- \times F^* \mid w \notin F \prec K\} \\
&\cup \{wF^0 \times F^* \mid w \in F,\ F \textit{ is an unbounded face of } K\} \\
&\cup \{wF^- \times G^* \mid w \in G,\ w \notin F \lhd G,\ G \leq K\},
\end{aligned} \tag{3.11}
$$

and

$$
\begin{aligned}
|\partial\mathcal{M}| = &\left(\bigcup [\{w\}\times\{v\}^* \mid v \textit{ is a vertex of } K,\ v \neq w]\right) \\
&\cup \left(\bigcup [\{w\}\times (F^* + \langle h\rangle) \mid F \textit{ is an extreme ray of } K]\right) \\
&\cup \left(\bigcup [F \times F^* \mid \{w\} \neq F \leq K]\right).
\end{aligned} \tag{3.12}
$$

Note that

$$
|\partial\mathcal{M}| \supset (\{w\}\times (R^n \backslash \{w\}^*)) \cup \left(\bigcup [F \times F^* \mid \{w\} \neq F \leq K]\right).
$$

4. Path-following technique. Let $\mathcal{M}$ be the $(n+1)$-dimensional subdivided manifold obtained from the GPDM $(\mathcal{P}, \mathcal{D}; d)$ as described in the previous section, and let f be a continuously differentiable function from K into R^n. To find a stationary point of f on K, we consider the system

$$
g(x, y) \equiv f(x) + y = 0, \qquad (x, y) \in |\mathcal{M}|. \tag{4.1}
$$

If $0 \in R^n$ is a regular value of the mapping g, then from applying Theorem 2.1 to system (4.1) we obtain that $g^{-1}(0)$ consists of disjoint paths and loops. Suppose the starting point w is not a stationary point of f on K. Then we see from Lemma 3.2 that $(w, -f(w)) \in g^{-1}(0) \cap |\partial\mathcal{M}|$. Consequently, the connected component of $g^{-1}(0)$ containing $(w, -f(w))$ is a path. In the following, we denote this path by S. Also, according to Theorem 2.1, if the path S is bounded, then it will provide a distinct end point (x, y) in $|\partial\mathcal{M}|$. Since (x, y) satisfies the system of equations (4.1), $y = -f(x)$. If $x = w$, (x, y) would coincide with $(w, -f(w))$. Therefore, according to (3.12), $(x, y) = (x, -f(x))$ lies in $F \times F^*$ for some face F of K and x is a stationary point of f on K.

To follow the path S in $|\mathcal{M}|$, we subdivide K into simplices such that each cell X in $\bar{\mathcal{P}}$ is triangulated. An appropriate simplicial subdivision of K is obtained by first triangulating the set K^- as described in [7]. Note that the starting point w is a vertex of this triangulation. In order to triangulate K^+, note that K^+ is the union of $K^0 + \langle h\rangle$ and $F^+ + \langle h\rangle$ over all unbounded facets F of K. The subset $K^0 + \langle h\rangle$ can be triangulated in exactly the same way as wK^0; and each subset $F^+ + \langle h\rangle$ can be triangulated in a similar way as wF^- by using projections of $w + h$ on the faces of F^+ instead of projections of w on the faces of F^-, as illustrated in Fig. 4.1.

Let $\bar{f}$ be the piecewise linear approximation of f with respect to the triangulation. Taking $\bar{f}$ instead of f in (4.1), the path T of solutions to (4.1) originating at $(w, -f(w)) = (w, -\bar{f}(w))$ is piecewise linear and can therefore be followed by making pivoting steps in subsequent systems of linear equations. For ease of description we restrict ourselves

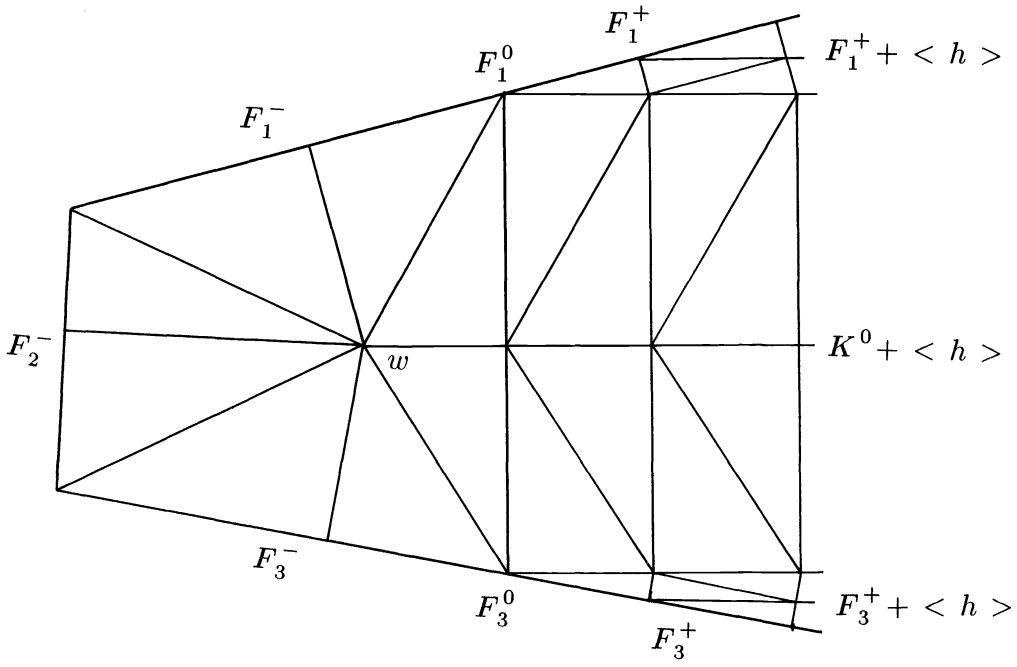


FIG. 4.1

to a polyhedron K for which none of the inequalities $(a_i)^t x \leqq b_i$ is redundant and each vertex is an end point of exactly n one-faces of K. To start the algorithm we first solve the linear program

$$\begin{aligned} &\min && f(w)^t x \\ &\text{s.t.} && A^t x \leqq b \\ &&& h^t x \leqq h_0. \end{aligned} \tag{4.2}$$

By the choice of h and h_0, this problem always has an optimal solution, which is some vertex, for instance v, of the feasible region. If the constraint $h^t x \leqq h_0$ is not binding at v, v is a vertex of K itself and so we take $F = \{v\}$ and find a one-dimensional simplex σ of the triangulation in $wF = wF^-$ which has w as a vertex. Let us denote w by w^1 and the other vertex of σ by w^2. Let μ_k^* be a dual optimal solution of (4.2) corresponding to the kth constraint $(a_k)^t x \leqq b_k$. Then, barring degeneracy $\mu_k^* > 0$, if and only if the kth constraint is binding at v, i.e., $k \in I(F)$. Then we see that $(\lambda_1^*, \lambda_2^*) = (1, 0)$ and μ_k^*, $k \in I(F)$ satisfy the system of linear equations

$$\sum_{i=1}^{2} \lambda_i \begin{bmatrix} f(w^i) \\ 1 \end{bmatrix} + \sum_{k \in I(F)} \mu_k \begin{bmatrix} a_k \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

When the constraint $h^t x \leqq h_0$ is binding at v, v lies on an extreme ray of K. We take the ray as F and find a one-dimensional simplex σ containing w as a vertex in wF^0. Let μ_k^*, $k \in I(F)$, and α^* be a dual optimal solution of (4.2) corresponding to the kth constraint $(a_k)^t x \leqq b_k$ and the last constraint $h^t x \leqq h_0$, respectively. Then we see that $(\lambda_1^*, \lambda_2^*) = (1, 0)$, μ_k^*, $k \in I(F)$, and α^* are a solution of

$$\sum_{i=1}^{2} \lambda_i \begin{bmatrix} f(w^i) \\ 1 \end{bmatrix} + \sum_{k \in I(F)} \mu_k \begin{bmatrix} a_k \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} h \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

In both cases we leave the starting point w by increasing λ_2 from zero. Note that (x, y), which is $(\sum_{i=1}^{2} \lambda_i w^i, \sum_{k\in I(F)} \mu_k a_k)$ and $(\sum_{i=1}^{2} \lambda_i w^i, \sum_{k\in I(F)} \mu_k a_k + \alpha h)$, respectively, is a point on the path T as long as all variables remain nonnegative.

Now, in general, let (x, y) be a point on the path T. Then in some t-cell X of $\bar{\mathcal{P}}$ there is a simplex σ with vertices $w^1, \cdots, w^{t+1}$ such that x lies in σ and $\bar{f}(x)$ in X^d. Hence, there exist nonnegative numbers λ_i^*, $i = 1, \cdots, t+1$, such that $x = \sum_i \lambda_i^* w^i$ and $\sum_i \lambda_i^* = 1$. Moreover, if $X = wF^-$, there exist nonnegative numbers μ_k^*, $k \in I(F)$, such that $y = \sum_{k\in I(F)} \mu_k^* a_k$; and if $X = wF^0$ or $X = F^+$, there exist nonnegative numbers μ_k^*, $k \in I(F)$, and α^* such that $y = \sum_{k\in I(F)} \mu_k^* a_k + \alpha^* h$. In case not all vertices of K are determined by n one-faces of K, we refer to [7]. Since (x, y) is a solution of (4.1) with $\bar{f}$ instead of f and $\bar{f}(x) = \sum_i \lambda_i^* f(w^i)$, it follows that λ_i^*, $i = 1, \cdots, t+1$, μ_k^*, $k \in I(F)$, is a nonnegative solution to the system of linear equations

$$\sum_{i=1}^{t+1} \lambda_i \begin{bmatrix} f(w^i) \\ 1 \end{bmatrix} + \sum_{k\in I(F)} \mu_k \begin{bmatrix} a_k \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.3}$$

if $X = wF^-$, and that λ_i^*, $i = 1, \cdots, t+1$, μ_k^*, $k \in I(F)$, α^* is a nonnegative solution to the system of linear equations

$$\sum_{i=1}^{t+1} \lambda_i \begin{bmatrix} f(w^i) \\ 1 \end{bmatrix} + \sum_{k\in I(F)} \mu_k \begin{bmatrix} a_k \\ 0 \end{bmatrix} + \alpha \begin{bmatrix} h \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4.4}$$

if $X = wF^0$ or $X = F^+$. The system (4.3) or (4.4) has a line segment of solutions corresponding to a line segment of points $x = \sum_i \lambda_i w^i$ in σ, assuming nondegeneracy. At an end point of solutions one of the variables is equal to zero. When $\lambda_i = 0$ for some $i \in \{1, \cdots, t+1\}$, x lies in the facet τ opposite the vertex w^i of σ. This facet lies either in the boundary of X or is a facet of just one other t-simplex $\bar{\sigma}$ in the cell X with vertices w^j, $j \neq i$, and $\bar{w}^i \neq w^i$. Then in the latter case, to continue the path T in $\bar{\sigma}$, a pivoting step is made with $(f(\bar{w}^i)^t, 1)^t$. Suppose τ lies in the boundary of X and $X = wF^-$. Then x is a stationary point of $\bar{f}$ on K if τ lies in F^-. Otherwise, either τ lies in wG^- with G a facet of F or τ lies in wF^0. In the first case the path T can be continued in wG^- by pivoting $((a_k)^t, 0)^t$ into (4.3), where k is the unique index in $I(G)$, not in $I(F)$. In the latter case the path T can be continued in wF^0 by pivoting $(h^t, 0)^t$ into (4.3). Now, suppose τ lies in the boundary of X and $X = wF^0$ or $X = F^+$. Then, when $X = wF^0$, τ lies either in wG^0 for some facet G of F or in F^0; and when $X = F^+$, τ lies either in G^+ for some facet G of F or in F^0. When τ is in wG^0 or G^+, the path T can be continued by pivoting $((a_k)^t, 0)^t$ into (4.4), where k is the unique index in $I(G)$, not in $I(F)$. When τ lies in F^0, τ is the facet of a unique t-simplex $\bar{\sigma}$ in F^+ if $X = wF^0$ and in wF^0 if $X = F^+$; and the path T can be continued in $\bar{\sigma}$ by making a pivoting step with $(f(\bar{w})^t, 1)^t$ in (4.4), where $\bar{w}$ is the vertex of $\bar{\sigma}$ opposite τ.

We now consider the case that at an end point of solutions of (4.3) or (4.4) we have that $\mu_k = 0$ for some $k \in I(F)$. Let G be the unique face of K such that $I(G) = I(F)\backslash\{k\}$. Then with $x = \sum_i \lambda_i w^i$ we have that $\bar{f}(x) = \sum_i \lambda_i f(w^i) = -\sum_{k\in I(G)} \mu_k a_k \in G^*$. First, suppose that $X = wF^-$. If $w \in G$, then also $x \in G$, since F is a facet of G. Therefore, $w \in G$ implies $(x, -f(x)) \in G \times G^*$, and hence x is a stationary point of $\bar{f}$ on K. In case $w \notin G$ or if $X = wF^0$ or F^+, then σ is a facet of a unique $(t+1)$-dimensional simplex $\bar{\sigma}$ in wG^-, wG^0, or G^+, respectively; and the path can be continued in $\bar{\sigma}$ by making a pivoting step with $(f(\bar{w})^t, 1)^t$ in (4.3) or (4.4), where $\bar{w}$ is the vertex of $\bar{\sigma}$ opposite σ.

Finally, we consider the case that in (4.4), $\alpha = 0$ at an end point. If $X = wF^0$ and $w \notin F$, then σ is a facet of a unique $(t+1)$-simplex $\bar{\sigma}$ in wF^- and the path can be continued in wF^- by making a pivoting step with $(f(\bar{w})^t, 1)^t$ in (4.4), where $\bar{w}$ is the

vertex of $\bar{\sigma}$ opposite σ. If $X = wF^0$ and $w \in F$ or if $X = F^+$, then $x = \sum_i \lambda_i w^i \in F$ and $\bar{f}(x) = \sum_i \lambda_i f(w^i) = -\sum_{k \in I(F)} \mu_k a_k \in F^*$, so that x is a stationary point of $\bar{f}$ on K.

This completes the description of how to follow approximately the path S by making alternating pivoting and replacement steps for a sequence of adjacent simplices of varying dimension. When this sequence does not diverge and terminates with a simplex, it contains a stationary point $\bar{x}$ of $\bar{f}$ on K. This point $\bar{x}$ is an approximate stationary point of f on K. To improve the accuracy of the approximation, if necessary, we can take a finer triangulation of K with the point $\bar{x}$ as the new starting point w and apply the same procedure.

5. Convergence condition. In this section we state a condition under which the path S is bounded and therefore leads from w to a stationary point of f on K.

LEMMA 5.1. *Let (x, y) be a solution of the system*

$$g(x, y) = 0, \qquad (x, y) \in F^+ \times (F^* + \langle h \rangle). \tag{5.1}$$

If x is not a stationary point, then

$$r^t y > 0$$

for any nonzero vector r in the cone C such that $(a_i)^t r = 0$ for all $i \in I(F)$.

Proof. The point y in $F^* + \langle h \rangle$ is equal to $B\mu + \alpha h$ for some vector $\mu \geqq 0$ and number $\alpha \geqq 0$, where B denotes the submatrix of A consisting of the column vectors a_i for $i \in I(F)$. Since x is not a stationary point, $\alpha > 0$. Then

$$r^t y = r^t(B\mu + \alpha h) = (B^t r)^t \mu + \alpha h^t r = \alpha h^t r > 0$$

by the choice of h. □

CONDITION 5.2. *There is a set $U \subset R^n$ such that $U \cap K$ is bounded and for each point $x \in K \setminus U$ there is a nonzero vector $\tilde{r}$ in $C \cap \{r \in R^n \mid (a_i)^t r = 0$ if $(a_i)^t x = b_i\}$ satisfying*

$$\tilde{r}^t f(x) \geqq 0.$$

LEMMA 5.3. *Under Condition 5.2 the path S does not diverge.*

Proof. Suppose the contrary. Then there is a solution $(\tilde{x}, \tilde{y})$ of the system (4.1) such that $\tilde{x} \in F^+ \setminus U$ for some face F, since the continuity of the function f requires the x-component to diverge. Therefore, by Lemma 5.1 and Condition 5.2, we see that $\tilde{r}^t(f(\tilde{x}) + \tilde{y}) > 0$ for some vector $\tilde{r}$, which contradicts the statement that $(\tilde{x}, \tilde{y})$ is a solution of (4.1). □

6. Stationary point problems on a Cartesian product of a polytope and a polyhedron. We consider a stationary point problem defined on the Cartesian product of a polytope and a polyhedron. The product is again a polyhedron and the discussion of §§ 3, 4, and 5 could still be applied to this case. However, it will be quite useful to consider it separately because a lot of problems are defined on such product sets. Let

$$K_1 = \{x_1 \in R^{n_1} \mid A_1^t x_1 \leqq b_1\}$$

be a nonempty polytope and let

$$K_2 = \{x_2 \in R^{n_2} \mid A_2^t x_2 \leqq b_2\}$$

be a nonempty, convex, unbounded, and pointed polyhedron, with A_i an $n_i \times m_i$ matrix and b_i an m_i-vector for $i = 1, 2$. We consider the stationary point problem for a continuous function f from $K_1 \times K_2$ to $R^{n_1} \times R^{n_2}$. We denote $f(x)$ by $f(x) = (f_1(x_1, x_2), f_2(x_1, x_2))$. Then $(x_1, x_2) \in K_1 \times K_2$ is a stationary point of f on $K_1 \times K_2$ if

$$(z_1 - x_1)^t f_1(x_1, x_2) + (z_2 - x_2)^t f_2(x_1, x_2) \geqq 0$$

for any point $(z_1, z_2) \in K_1 \times K_2$.

In the same way as in the preceding sections, we will construct a GPDM by introducing an artificial hyperplane and corresponding halfspaces defined by

$$H^{\pi} = \{(x_1, x_2) \in R^{n_1+n_2} \mid h_2^t x_2 \, \rho \, h_0\},$$

where π is $-$, 0, and $+$ when ρ is $\leqq$, $=$, and $\geqq$, respectively. $h_2 = -A_2\gamma$ for some fixed positive vector γ, and $h_0 > 0$ is chosen such that the interior of the halfspace H^- contains all vertices of $K_1 \times K_2$ as well as the starting point $w = (w_1, w_2)$. Note that $h_2^t r_2 > 0$ for any nonzero vector r_2 in the set

$$C_2 = \{r_2 \in R^{n_2} \mid A_2^t r_2 \leqq 0\}$$

of directions of rays of K_2, which we have seen is a pointed cone. It is clear that a face F of $K_1 \times K_2$ is itself a Cartesian product of faces of K_1 and K_2, which we will denote by F_1 and F_2, respectively. Let

$$H_2^{\pi} = \{x_2 \in R^{n_2} \mid h_2^t x_2 \, \rho \, h_0\},$$

where π is $-$, 0, and $+$ when ρ is $\leqq$, $=$, and $\geqq$, respectively. We define

$$F_2^{\pi} = F_2 \cap H_2^{\pi} \quad \text{for} \quad \pi = -, 0, \text{and} +.$$

Then

$$F^{\pi} = F_1 \times F_2^{\pi} \quad \text{for} \quad \pi = -, 0, \text{and} +.$$

It is also clear that the normal cone F^* corresponding to a face F of K is given by

$$F^* = F_1^* \times F_2^*,$$

where F_1^* and F_2^* are defined with respect to $K_1 \subset R^{n_1}$ and $K_2 \subset R^{n_2}$, respectively. Thus, with the dual operator d defined as follows, we obtain a GPDM:

$$\begin{aligned}
&(w(F_1 \times F_2^-))^d = F_1^* \times F_2^* && \text{if } w \notin F_1 \times F_2 < K;\\
&(w(F_1 \times F_2^0))^d = F_1^* \times (F_2^* + \langle h_2 \rangle) && \text{if } F_2 \text{ is an unbounded face of } K_2;\\
&(F_1 \times F_2^+)^d = F_1^* \times (F_2^* + \langle h_2 \rangle) && \text{if } F_2 \text{ is an unbounded face of } K_2;\\
&(F_1 \times F_2^-)^d = \varnothing && \text{if } w \notin F_1 \times F_2 < K;\\
&(F_1 \times F_2^0)^d = \varnothing && \text{if } F_2 \text{ is an unbounded face of } K_2;\\
&(\{w\})^d = \varnothing.
\end{aligned}$$

The collection $\mathcal{M}$ of cells, each cell being the Cartesian product of a primal cell and its dual, is clearly a subdivided (n_1+n_2+1)-manifold. The boundary $\partial\mathcal{M}$ contains $\{(w_1, w_2)\} \times (R^{n_1} \times R^{n_2} \backslash \{(w_1, w_2)\}^*)$. It also contains $(F_1 \times F_2) \times (F_1^* \times F_2^*)$ for all faces F_1 of K_1 and for all faces F_2 of K_2 when (w_1, w_2) is not a vertex of K; and when (w_1, w_2) is a vertex of K, it contains $(F_1 \times F_2) \times (F_1^* \times F_2^*)$ for all faces F_1 of K_1 not equal to $\{w_1\}$ and for all faces F_2 of K_2 not equal to $\{w_2\}$. Therefore, when the starting point $w = (w_1, w_2)$ is not a stationary point, the point $(x_1, x_2, y_1, y_2) = (w_1, w_2, -f_1(w_1, w_2), -f_2(w_1, w_2))$ lies in the boundary $|\partial\mathcal{M}|$ of $|\mathcal{M}|$ and under the regular value assumption there is a path leading from it to either a stationary point or to infinity. Thus, in exactly the same way as in the preceding sections, the problem is now reduced to tracing the path S of solutions to the system

$$(f_1(x_1, x_2), f_2(x_1, x_2)) + (y_1, y_2) = 0, \qquad (x_1, x_2, y_1, y_2) \in |\mathcal{M}|.$$

The remarkable feature of this path is shown in the following lemma, where

$$S_x = \{x = (x_1, x_2) \mid (x_1, x_2, y_1, y_2) \in S \text{ for some } (y_1, y_2) \in R^{n_1+n_2}\}.$$

LEMMA 6.1. *If* $(\tilde{x}_1, \tilde{x}_2) \in S_x \cap H^+$, *then* $\tilde{x}_1$ *is a stationary point for* $f_1(\cdot, \tilde{x}_2)$ *on* K_1, *i.e.*, $\tilde{x}_1^t f_1(\tilde{x}_1, \tilde{x}_2) \leqq x_1^t f_1(\tilde{x}_1, \tilde{x}_2)$ *for all* $x_1 \in K_1$.

Proof. Since $(\tilde{x}_1, \tilde{x}_2) \in H^+$, it is in $F_1 \times F_2^+$ for some face F_1 of K_1 and some unbounded face F_2 of K_2. By the construction of the GPDM,

$$(-f_1(\tilde{x}_1, \tilde{x}_2), -f_2(\tilde{x}_1, \tilde{x}_2)) \in F_1^* \times (F_2^* + \langle h_2 \rangle).$$

This means that $\tilde{x}_1$ is a stationary point for $f_1(\cdot, \tilde{x}_2)$ on K_1. □

LEMMA 6.2. *Let* $(\tilde{x}_1, \tilde{x}_2, \tilde{y}_1, \tilde{y}_2)$ *be a point of* S. *Suppose that* $(\tilde{x}_1, \tilde{x}_2)$ *is not a stationary point and* $\tilde{x}_2$ *lies in* H_2^+. *Then*

$$r_2^t \tilde{y}_2 > 0$$

for any nonzero vector r_2 *in the cone* C_2 *such that* $(a_{2i})^t r_2 = 0$ *whenever* $(a_{2i})^t \tilde{x}_2 = b_{2i}$, *where* a_{2i} *is the* i*th column of* A_2 *and* b_{2i} *is the* i*th component of* b_2.

Proof. Let B_2 be the submatrix of A_2 consisting of the columns a_{2i} such that $(a_{2i})^t \tilde{x}_2 = b_{2i}$. Then $\tilde{y}_2 = B_2\mu + \alpha h_2$ for some vector $\mu \geqq 0$ and real number $\alpha \geqq 0$. Since $(\tilde{x}_1, \tilde{x}_2)$ is in H^+ and is not a stationary point, we have $\alpha > 0$ by Lemma 6.1. Therefore,

$$r_2^t \tilde{y}_2 = r_2^t (B_2\mu + \alpha h_2) = \alpha h_2^t r_2 > 0. \qquad \square$$

CONDITION 6.3. *There is a set* $U_2 \subset R^{n_2}$ *such that* $U_2 \cap K_2$ *is bounded and for each point* $\tilde{x}_2 \in K_2 \setminus U_2$ *one of the following conditions holds*:

(1) *there is no stationary point for* $f_1(\cdot, \tilde{x}_2)$ *on* K_1,

(2) *for each point* $x_1 \in K_1$, *there is a nonzero vector* $\tilde{r}_2$ *in* $C_2 \cap \{r_2 \in R^{n_2} | (a_{2i})^t r_2 = 0$ *if* $(a_{2i})^t \tilde{x}_2 = b_{2i}\}$ *such that* $\tilde{r}_2^t f_2(x_1, \tilde{x}_2) \geqq 0$.

THEOREM 6.4. *Under Condition* 6.3 *the path* S *does not diverge.*

Proof. Suppose the contrary. Then there is a point $(\tilde{x}_1, \tilde{x}_2) \in S_x \cap H^+$ such that $\tilde{x}_2 \notin U_2$. By Lemma 6.1, $\tilde{x}_1$ is a stationary point for $f_1(\cdot, \tilde{x}_2)$ on K_1. Therefore, condition (2) must be satisfied at this point, so that for some nonzero vector $\tilde{r}_2$ in $C_2 \cap \{r_2 \in R^{n_2} | (a_{2i})^t r_2 = 0$ if $(a_{2i})^t \tilde{x}_2 = b_{2i}\}$ we must have

$$\tilde{r}_2^t f_2(\tilde{x}_1, \tilde{x}_2) \geqq 0.$$

On the other hand, we have seen in Lemma 6.2 that

$$\tilde{r}_2^t f_2(\tilde{x}_1, \tilde{x}_2) = \tilde{r}_2^t (-\tilde{y}_2) < 0.$$

This is a contradiction. □

7. Linear stationary point problems. In this section, we consider a special but important case where the function f from K to R^n is an affine function, i.e., $f(x) = Qx + q$, where Q is an $n \times n$ matrix and q is an n-vector. For simplicity of notation we confine ourselves to the linear stationary point problem defined on a polyhedron instead of the product of a polytope and a polyhedron. As for complementary pivoting algorithms for solving a linear complementarity problem, we show that if the matrix Q is copositive plus on the polyhedral cone C and the problem has a stationary point, the path does not go to infinity and consequently leads to one of the stationary points.

DEFINITION 7.1. The matrix Q is copositive plus on C if

(1) $r^t Q r \geqq 0$ for any $r \in C$,

(2) $(Q + Q^t) r = 0$ if $r \in C$ and $r^t Q r = 0$.

LEMMA 7.2. *There exists no point* $x \in K$ *such that* $Qx + q = -A\mu$ *for some vector* $\mu \geqq 0$ *if and only if there is a* $(v, u) \in R^n \times R^m$ *such that* $v \in C$, $Q^t v = Au$, $b^t u + q^t v < 0$, *and* $u \geqq 0$.

Proof. There exists no point x in K satisfying $Qx+q=-A\mu$ for some $\mu \geqq 0$ if and only if the system

$$A^t(x_1-x_2) \leqq b, \qquad Q(x_1-x_2)+q=-A\mu, \qquad x_1, x_2, \mu \geqq 0 \tag{7.1}$$

is not solvable. By Farkas' Alternative Theorem, we have an equivalent statement to (7.1): the following system:

$$Q^tv-Au=0, \qquad A^tv \leqq 0, \qquad u \geqq 0, \qquad b^tu+q^tv<0 \tag{7.2}$$

is solvable. This means the existence of a point v in C such that $Q^tv=Au$ and $b^tu+q^tv<0$ for some $u \geqq 0$. □

LEMMA 7.3. *Let Q be copositive plus on C. If the path S is unbounded and does not contain a point which provides a stationary point, then there are no stationary points.*

Proof. Suppose S is unbounded, then there are $(x, y) \in S$ and $(\bar{x}, \bar{y}) \neq 0$ such that $(x, y)+\beta(\bar{x}, \bar{y}) \in S$ for any $\beta \geqq 0$. Then

$$\bar{y}+Q\bar{x}=0. \tag{7.3}$$

Moreover, as β increases, $(x, y)+\beta(\bar{x}, \bar{y})$ will be entirely contained in a cell $F^+ \times (F^*+\langle h \rangle)$ for some face $F \trianglelefteq K$. Here note that $\bar{x} \neq 0$ because the contrary would yield $(\bar{x}, \bar{y})=0$. Then we have

$$x \in F^+$$

$$y=y'+\lambda h \quad \text{for some } y' \in F^* \text{ and some } \lambda \geqq 0,$$

and

$$\bar{x} \in C \cap \{r \in R^n \mid (a_i)^t r=0 \text{ if } (a_i)^t x=b_i\}$$

$$\bar{y}=\bar{y}'+\mu h \quad \text{for some } \bar{y}' \in F^* \quad \text{and some } \mu \geqq 0.$$

Therefore, we have

$$\bar{x}^tQ\bar{x}=\bar{x}^t(-\bar{y})=\bar{x}^t(-\bar{y}'-\mu h)=-\mu\bar{x}^th.$$

Suppose $\mu>0$. By the choice of h and since $\bar{x} \in C$, we have $\mu\bar{x}^th>0$, which contradicts that Q is copositve plus on C. Therefore, $\mu=0$ and $\bar{x}^tQ\bar{x}=0$. If $\lambda=0$, then $y=y' \in F^*$. This means that the point x is a stationary point. Since we have assumed that S does not contain such a point, we see that $\lambda>0$. Since $\bar{x}^tQ\bar{x}=0$ implies $(Q+Q^t)\bar{x}=0$, we have

$$Q^t\bar{x}=-Q\bar{x}=\bar{y}=\bar{y}'+\mu h=\bar{y}' \in F^*.$$

In other words, there is some vector u satisfying

$$\begin{aligned} &Q^t\bar{x}=Au, \\ &u_i \geqq 0 \quad \text{for } i \in I(F), \\ &u_i=0 \quad \text{for } i \notin I(F). \end{aligned} \tag{7.4}$$

We also have

$$\bar{x}^t(-y)=\bar{x}^t(Qx+q)=x^tQ^t\bar{x}+q^t\bar{x}=x^t(-Q\bar{x})+q^t\bar{x}=x^t\bar{y}+q^t\bar{x}. \tag{7.5}$$

On the other hand, since $\bar{x} \in C \cap \{r \in R^n \mid (a_i)^t r=0 \text{ if } (a_i)^t x=b_i\}$ and $y' \in F^*$,

$$\bar{x}^t(-y)=\bar{x}^t(-y'-\lambda h)=-\bar{x}^ty'-\lambda\bar{x}^th=-\lambda\bar{x}^th<0. \tag{7.6}$$

From (7.5) and (7.6) we have $x'\bar{y}+q'\bar{x}<0$. Since $x\in F^+$, we also have that $A'x+s=b$ for some slack variable vector s satisfying

$$s_i \geqq 0 \quad \text{for } i \notin I(F),$$

$$s_i = 0 \quad \text{for } i \in I(F).$$

Then

$$\begin{aligned} b'u+q'\bar{x} &< b'u-x'\bar{y}=(A'x+s)'u-x'(-Q\bar{x}) \\ &= x'Au+s'u-x'(Q'\bar{x})=x'(Au-Q'\bar{x})+s'u=0. \end{aligned} \tag{7.7}$$

From (7.4), (7.7), and Lemma 7.2, we conclude that there are no stationary points. □

The algorithm for tracing the piecewise linear path S, being linear on each cell of $\mathcal{M}$, is quite similar to that proposed in Yamamoto [8] for solving linear stationary point problems on polytopes. We will only give an outline here. Suppose we are at a point (x, y) on the path, i.e.,

$$Qx+q+y=0, \qquad (x, y)\in X\times X^d, \tag{7.8}$$

for some cell $X\times X^d$ of $\mathcal{M}$. By the decomposition theorem of a polyhedron, each point of a polyhedron is a sum of two points: a convex combination of vertices of the polyhedron and a nonnegative combination of directions of extreme rays. Let U and R be the sets of vertices and extreme rays of X, respectively. Then a point $x\in X$ is written as

$$x=\sum_{u\in U}\lambda_u u+\sum_{r\in R}\alpha_r r,$$

$$\sum_{u\in U}\lambda_u=1,$$

$$\lambda_u\geqq 0, \qquad \alpha_r\geqq 0.$$

On the other hand, X^d is the cone generated by coefficient vectors a_i of binding constraints of the face corresponding to X and the vector h. Then a point $y\in X^d$ is written as

$$y=\sum \mu_i a_i, \qquad \mu_i\geqq 0$$

if we denote h by a_0. Therefore, (7.8) has a solution if and only if the system

$$\begin{gathered} \sum\lambda_u\begin{bmatrix}Qu\\1\end{bmatrix}+\sum\alpha_r\begin{bmatrix}Qr\\0\end{bmatrix}+\sum\mu_i\begin{bmatrix}a_i\\0\end{bmatrix}=\begin{bmatrix}-q\\1\end{bmatrix}, \\ \lambda_u\geqq 0, \qquad \alpha_r\geqq 0, \qquad \mu_i\geqq 0 \end{gathered} \tag{7.9}$$

has a solution (λ, α, μ). It should be noted here that a vertex of X is either the starting point w or a vertex of some face of K^- corresponding to X and that an extreme ray of X is also an extreme ray of some face of K. More precisely,

$$U=\{w\}\cup\{\text{vertices of } F^-\}, \qquad R=\varnothing \quad \text{when } X=wF^-,$$

$$U=\{w\}\cup\{\text{vertices of } F^0\}, \qquad R=\varnothing \quad \text{when } X=wF^0,$$

$$U=\{\text{vertices of } F^0\}, \qquad R=\{\text{extreme rays of } F\} \quad \text{when } X=F^+.$$

In every case a vertex or an extreme ray can be generated in need when we keep in storage the index set of binding constraints, including $H^0=\{x\in R^n \mid h'x=h_0\}$, determining the face F.

Suppose we are at an end point of the line segment or halfline of the path within $X \times X^d$. Since the path is linear within $X \times X^d$, an appropriate choice of the objective function $c_x x + c_y y$ makes the current end point the unique maximal solution of the linear program:

$$\begin{aligned} \max \quad & c_x x + c_y y, \\ \text{s.t.} \quad & x = \sum \lambda_u u + \sum \alpha_r r, \\ & y = \sum \mu_i a_i, \\ & \sum \lambda_u \begin{bmatrix} Qu \\ 1 \end{bmatrix} + \sum \alpha_r \begin{bmatrix} Qr \\ 0 \end{bmatrix} + \sum \mu_i \begin{bmatrix} a_i \\ 0 \end{bmatrix} = \begin{bmatrix} -q \\ 1 \end{bmatrix}, \\ & \lambda_u \geqq 0, \qquad \alpha_r \geqq 0, \qquad \mu_i \geqq 0. \end{aligned}$$

In fact, the outward normal vector at the point (x, y) to $X \times X^d$ may serve as (c_x, c_y). Then the other end point, when the path within $X \times X^d$ is a line segment, or the diverging direction, when it is a halfline, can be found by solving the following linear minimization program:

$$\begin{aligned} \min \quad & c_x x + c_y y, \\ \text{s.t.} \quad & x = \sum \lambda_u u + \sum \alpha_r r, \\ & y = \sum \mu_i a_i, \\ & \sum \lambda_u \begin{bmatrix} Qu \\ 1 \end{bmatrix} + \sum \alpha_r \begin{bmatrix} Qr \\ 0 \end{bmatrix} + \sum \mu_i \begin{bmatrix} a_i \\ 0 \end{bmatrix} = \begin{bmatrix} -q \\ 1 \end{bmatrix}, \\ & \lambda_\mu \geqq 0, \qquad \alpha_r \geqq 0, \qquad \mu_i \geqq 0. \end{aligned}$$

From this we see that solving the problem is a typical application of the Dantzig-Wolfe decomposition principle for large structured linear programs. By solving a sequence of these problems we can trace the path and finally, after a finite number of iterations, we meet with an end point of the path or find that the path goes to infinity.

REFERENCES

[1] Y. DAI AND Y. YAMAMOTO, *The path following algorithm for stationary point problems on polyhedral cones*, J. Oper. Res. Soc. Japan, 32 (1989), pp. 286-309.

[2] M. KOJIMA, *An introduction to variable dimension algorithms for solving systems of equations*, in Numerical Solution of Nonlinear Equations, Springer-Verlag, Berlin, New York, 1981, pp. 199-237.

[3] S. KARAMADIAN, *Generalized complementarity problem*, J. Optim. Theory Appl., 8 (1971), pp. 161-168.

[4] M. KOJIMA AND Y. YAMAMOTO, *Variable dimension algorithms: Basic theory, interpretations and extensions of some existing methods*, Math. Programming, 24 (1982), pp. 177-215.

[5] R. SAIGAL, *Extension of the generalized complementarity problem*, Math. Oper. Res., 1 (1976), pp. 260-266.

[6] J. STOER AND C. WITZGALL, *Convexity and Optimization in Finite Dimensions* I, Springer-Verlag, Berlin, New York, 1978.

[7] A. J. J. TALMAN AND Y, YAMAMOTO, *A simplicial algorithm for stationary point problems*, Math. Oper. Res., 14 (1989), pp. 383-399.

[8] Y. YAMAMOTO, *A path following algorithm for stationary point problems*, J. Oper. Res. Soc. Japan, 30 (1987), pp. 181-198.

[9] ———, *Stationary point problems and a path-following algorithm*, in Proc. 8th Mathematical Programming Symposium, Hiroshima, Japan, 1987, pp. 153-170.

[10] ———, *Fixed point algorithms for stationary point problems*, Mathematical Programming, Recent Developments and Applications, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1989, pp. 283-307.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 166-190, May 1991

002

CONES OF MATRICES AND SET-FUNCTIONS AND 0–1 OPTIMIZATION*

L. LOVÁSZ† AND A. SCHRIJVER‡

Abstract. It has been recognized recently that to represent a polyhedron as the projection of a higher-dimensional, but simpler, polyhedron, is a powerful tool in polyhedral combinatorics. A general method is developed to construct higher-dimensional polyhedra (or, in some cases, convex sets) whose projection approximates the convex hull of 0–1 valued solutions of a system of linear inequalities. An important feature of these approximations is that one can optimize any linear objective function over them in polynomial time.

In the special case of the vertex packing polytope, a sequence of systems of inequalities is obtained such that the first system already includes clique, odd hole, odd antihole, wheel, and orthogonality constraints. In particular, for perfect (and many other) graphs, this first system gives the vertex packing polytope. For various classes of graphs, including t-perfect graphs, it follows that the stable set polytope is the projection of a polytope with a polynomial number of facets.

An extension of the method is also discussed which establishes a connection with certain submodular functions and the Möbius function of a lattice.

Key words. polyhedron, cone, vertex packing polytope, perfect graph, Möbius function

AMS(MOS) subject classifications. 05C35, 90C10, 90C27

0. Introduction. One of the most important methods in combinatorial optimization is that which represents each feasible solution of the problem by a 0–1 vector (usually the incidence vector of the appropriate set), and then describes the convex hull K of the solutions by a system of linear inequalities. In the nicest cases (e.g., in the case of the bipartite matching problem) we obtain a system that has polynomial size (measured in the natural "size" n of the problem). In such a case, we can compute the maximum of any linear objective function in polynomial time by solving a linear program. In other cases, however, the convex hull of feasible solutions has exponentially many facets and so can only be described by a linear program of exponential size. For many combinatorial optimization problems (including those solvable in polynomial time), this exponentially large set of linear inequalities is still "nice" in one sense or another. We mention two possible notions of "niceness":

—Given an inequality in the system, there is a polynomial size certificate of the fact that it is valid for K. If this is the case, the problem of determining whether a given vector is in K is in the complexity class co-NP.

—There is a polynomial time separation algorithm for the system; that is, given a vector, we can check in polynomial time whether it satisfies the system, and if not, we can find an inequality in the system that is violated. It follows, then, from general results on the ellipsoid method (see Grötschel, Lovász, and Schrijver [14]) that every linear objective function can be optimized over K in polynomial time.

Many important theorems in combinatorial optimization provide such "nice" descriptions of polyhedra. Important examples of polyhedra with "nice" descriptions are matching polyhedra, matroid polyhedra, stable set polyhedra for perfect graphs, etc. On the other hand, stable set polyhedra, in general, or travelling salesman polyhedra, are not known to have "nice" descriptions (and probably do not have any). Typically, to find such a "nice" description and to prove its correctness, one needs ad

* Received by the editors January 10, 1990; accepted for publication (in revised form) October 18, 1990.

† Department of Computer Science, Eötvös Loránd University, Budapest, Hungary H-1088, and Department of Computer Science, Princeton University, Princeton, New Jersey 08544.

‡ Mathematische Centrum, Kruislaan 413, 1098 SJ Amsterdam, the Netherlands.

hoc methods depending on the combinatorial structure. However, one can mention two general ideas that can help in obtaining such linear descriptions:

—*Gomory-Chvátal cuts.* Let P be a polytope with integral vertices. Assume that we have already found a system of linear inequalities valid for P whose *integral* solutions are precisely the integral vectors in P. The solution set of this system is a polytope K containing P which will in general be larger than P. We can generate further linear inequalities valid for P (but not necessarily for K) as follows. Given a linear inequality

$$\sum_i a_i x_i \leqq \alpha$$

valid for K, where the a_i are integers, the inequality

$$\sum_i a_i x_i \leqq \lfloor \alpha \rfloor$$

is still valid for P but may eliminate some part of K. Gomory [11] used a special version of this construction in his integer programming algorithm. If we take all inequalities obtainable in this way, they define a polytope K' with $P \subseteq K' \subset K$. Repeating this with K' in place of K we obtain K'', etc. Chvátal [8] proved that in a finite number of steps, we obtain the polytope P itself.

Unfortunately, the number of steps needed may be very large; it depends not only on the dimension but also on the coefficients of the system with which we start. Another problem with this procedure is that there is no efficient way known to implement it algorithmically. In particular, even if we know how to optimize a linear objective function over K in polynomial time (say, K is given by an explicit, polynomial size linear program), and $K' = P$, we know of no general method to optimize a linear objective function over P in polynomial time.

—*Projection representation* (*new variables*). This method has received much attention lately. The idea is that a projection of a polytope may have more facets than the polytope itself. This remark suggests that even if P has exponentially many facets, we may be able to represent it as the projection of a polytope Q in higher (but still polynomial) dimension, having only a polynomial number of facets. Among others, Barahona [4]; Liu [16]; Ball, Liu, and Pulleyblank [3]; Maculan [19]; Balas and Pulleyblank [1], [2]; Barahona and Mahjoub [5]; and Cameron and Edmonds [6] have provided nontrivial examples of such a representation. It is easy to see that such a representation can be used to optimize linear objective functions over P in polynomial time. In the negative direction, Yannakakis [26] proved that the travelling salesman polytope and the matching polytope of complete graphs cannot be represented this way, assuming that the representation is "canonical." (Let $P \subseteq \mathbb{R}^n$ and $P' \subseteq \mathbb{R}^m$ be two polytopes. We say that a projection representation $\pi : P' \to P$ is *canonical* if the group Γ of isometries of $\mathbb{R}^n$ preserving P has an action as isometries of $\mathbb{R}^m$ preserving P' so that the projection commutes with these actions. Such a representation is obtained, e.g., when new variables are introduced in a "canonical" way—in the case of the travelling salesman polytope, this could mean variables assigned to edges or certain other subgraphs, and constraints on these new variables are derived from local properties. If we have to start with a reference orientation, or with specifying a root, then the representation obtained will not be canonical.) No negative results seem to be known without this symmetry assumption.

One way to view our results is to provide a general procedure to create such liftings. The idea is to extend the method of Grötschel, Lovász, and Schrijver [12] for finding maximum stable sets in perfect graphs to general 0-1 programs. We represent

a feasible subset not by its incidence vector v but by the matrix vv^T. This squares the number of variables, but in return we obtain two new powerful ways to write down linear constraints. Projecting back to the "usual" space, we obtain a procedure somewhat similar to the Gomory–Chvátal procedure: it "cuts down" a convex set K to a new convex set K' so that all 0–1 solutions are preserved. In contrast to the Gomory–Chvátal cuts, however, any subroutine to optimize a linear objective function over K can be used to optimize a linear objective function over K'. Moreover, repeating the procedure at most n times, we obtain the convex hull P of 0–1 vectors in K.

Our method is closely related to recent work of Sherali and Adams [22]. They introduce new variables for products of the original ones and characterize the convex hull, in this high-dimensional space, of vectors associated with 0–1 solutions of the original problem. In this way they obtain a sequence of relaxations of the 0–1 optimization problem, the first of which is essentially the N operator introduced in § 1 below. Further, members of the two sequences of relaxations are different but closely related; some of our results in § 3, in particular, formula (6) and Theorem 3.3, follow directly from their work.

This method is also related to (but different from) the recent work of Pemantle, Propp, and Ullman [20] on the tensor powers of linear programs.

In § 1, we describe the method in general, and prove its basic properties. Section 2 contains applications to the vertex packing problem, one of the best studied combinatorial optimization problems. It will turn out that our method gives in one step almost all of the known classes of facets of the vertex packing polytope. It will follow, in particular, that if a graph has the property that its stable set polytope is described by the clique, odd hole, and odd antihole constraints, then its maximum stable set can be found in polynomial time.

In § 3 we put these results in a wider context by raising the dimension even higher. We introduce exponentially many new variables; in this high-dimensional space, rather simple and elegant polyhedral results can be obtained. The main part of the work is to "push down" the inequalities to a low dimension and to carry out the algorithms using only a polynomial number of variables and constraints. It will turn out that the methods in § 1, as well as other constructions like TH (G), as described in Grötschel, Lovász, and Schrijver [13], [14], follow in a natural way.

1. Matrix cuts. In this section we describe a general construction for "lifting" a 0–1 programming problem in n variables to n^2 variables, and then projecting it back to the n-space so that cuts, i.e., tighter inequalities still valid for all 0–1 solutions, are introduced. It will be convenient to deal with homogeneous systems of inequalities, i.e., with convex cones rather than polytopes. Therefore we embed the n-dimensional space in $\mathbb{R}^{n+1}$ as the hyperplane $x_0 = 1$. (The 0th variable will play a special role throughout.)

One way to view our constructions is to generate *quadratic* inequalities valid for all 0–1 solutions. These may be viewed as homogeneous linear inequalities in the $\binom{n}{2} + n + 1$-dimensional space, and they define a cone there. (This space can be identified with the space of symmetric $(n+1) \times (n+1)$ matrices.) We then combine these quadratic inequalities to eliminate all quadratic terms in order to obtain linear inequalities not derivable directly. This corresponds to projecting the cone down the $n+1$-dimensional space.

1.a. The construction of matrix cones and their projections. Let K be a convex cone in $\mathbb{R}^{n+1}$. Let K^* be its polar cone, i.e., the cone defined by

$$K^* = \{u \in \mathbb{R}^{n+1} : u^T x \geqq 0 \text{ for all } x \in K\}.$$

We denote by K^o the cone spanned by all 0–1 vectors in K. Let Q denote the cone spanned by all 0–1 vectors $x \in \mathbb{R}^{n+1}$ with $x_0 = 1$. We are interested in determining K^o, and generally we may restrict ourselves to subcones of Q. We denote by e_i the ith unit vector, and set $f_i = e_0 - e_i$. Note that the cone Q^* is spanned by the vectors e_i and f_i. For any $(n+1)\times(n+1)$ matrix Y, we denote by $\bar{Y}$ the vector composed of the diagonal entries of Y.

Let $K_1 \subseteq Q$ and $K_2 \subseteq Q$ be convex cones. We define the cone $M(K_1, K_2) \subseteq \mathbb{R}^{(n+1)\times(n+1)}$ consisting of all $(n+1)\times(n+1)$ matrices $Y = (y_{ij})$ satisfying (i), (ii), and (iii) below (for motivation, the reader may think of Y as a matrix of the form xx^T, where x is a 0–1 vector in $K_1 \cap K_2$).

(i) Y is symmetric;

(ii) $\bar{Y} = Ye_0$, i.e., $y_{ii} = y_{0i}$ for all $1 \leqq i \leqq n$;

(iii) $u^T Yv \geqq 0$ holds for every $u \in K_1^*$ and $v \in K_2^*$.

Note that (iii) can be rewritten as

(iii′) $YK_2^* \subseteq K_1$.

We shall also consider a slightly more complicated cone $M_+(K_1, K_2)$, consisting of matrices Y satisfying the following condition, in addition to (i), (ii), and (iii):

(iv) Y is positive semidefinite.

From the assumption that K_1 and K_2 are contained in Q it follows that every $Y = (y_{ij}) \in M(K_1, K_2)$ satisfies $y_{ij} \geqq 0$, $y_{ij} \leqq y_{ii} = y_{0i} \leqq y_{00}$, and $y_{ij} \geqq y_{ii} + y_{jj} - y_{00}$.

These cones of matrices are defined by linear constraints and so their polars can also be expressed quite nicely. Let U_{psd} denote the cone of positive semidefinite $(n+1)\times(n+1)$ matrices (which is self-dual in the space U_{sym} of symmetric matrices), and U_{skew} the linear space of skew symmetric $(n+1)\times(n+1)$ matrices (which is the orthogonal complement of U_{sym}). Let U_1 denote the linear space of $(n+1)\times(n+1)$ matrices (w_{ij}), where $w_{0j} = -w_{jj}$ for $1 \leqq j \leqq n$, $w_{00} = 0$ and $w_{ij} = 0$ if $i \neq 0$ and $i \neq j$. Note that U_1 is generated by the matrices $f_i e_i^T$ $(i = 1, \cdots, n)$.

With this notation, we have, by definition,

$$M(K_1, K_2)^* = U_1 + U_{\text{skew}} + \text{cone}\,\{uv^T : u \in K_1^*, v \in K_2^*\},$$

and

$$M_+(K_1, K_2)^* = U_1 + U_{\text{skew}} + U_{\text{psd}} + \text{cone}\,\{uv^T : u \in K_1^*, v \in K_2^*\}.$$

Note that only the last term depends on the cones K_1 and K_2. In this term, it would be enough to let u and v run over extreme rays of K_1^* and K_2^*, respectively. So if K_1 and K_2 are polyhedral, then so is $M(K_1, K_2)$, and the number of its facets is at most the product of the numbers of facets of K_1 and K_2.

Note that U_{psd} and hence $M_+(K_1, K_2)$ will generally be nonpolyhedral.

We project down these cones from the $(n+1)\times(n+1)$-dimensional space to the $(n+1)$-dimensional space by letting

$$N(K_1, K_2) = \{Ye_0 : Y \in M(K_1, K_2)\} = \{\bar{Y} : Y \in M(K_1, K_2)\}$$

and

$$N_+(K_1, K_2) = \{Ye_0 : Y \in M_+(K_1, K_2)\} = \{\bar{Y} : Y \in M_+(K_1, K_2)\}.$$

Clearly, $M(K_1, K_2) = M(K_2, K_1)$ and so $N(K_1, K_2) = N(K_2, K_1)$ (and similarly for the "+" subscripts).

If $A \in \mathbb{R}^{(n+1)\times(n+1)}$ is a linear transformation mapping the cone Q onto itself, then clearly $M(AK_1, AK_2) = AM(K_1, K_2)A^T$. If $n \geqq 2$, then from $AQ = Q$ it easily follows that $A^T e_0$ is parallel to e_0, and hence $N(AK_1, AK_2) = AN(K_1, K_2)$. In particular, we can "flip" coordinates, replacing x_i by $x_0 - x_i$ for some $i \neq 0$.

If K_1 and K_2 are polyhedral cones, then so too are $M(K_1, K_2)$ and $N(K_1, K_2)$. The cones $M_+(K_1, K_2)$ and $N_+(K_1, K_2)$ are also convex (but generally not polyhedral), since (iv) is equivalent to an infinite number of linear inequalities.

LEMMA 1.1. $(K_1 \cap K_2)^o \subseteq N_+(K_1, K_2) \subseteq N(K_1, K_2) \subseteq K_1 \cap K_2$.

Proof. (1) Let x be any nonzero 0-1 vector in $K_1 \cap K_2$. Since $K_1 \subseteq Q$, we must have $x_0 = 1$. Using this it is easy to check that the matrix $Y = xx^T$ satisfies (i)-(iv). Hence $x = Ye_0 \in N_+(K_1, K_2)$.

(2) $N_+(K_1, K_2) \subseteq (K_1, K_2)$ trivially.

(3) Let $x \in N(K_1, K_2)$. Then there exists a matrix Y satisfying (i)-(iv) such that $x = Ye_0$. Now, by our hypothesis that $K_1 \subseteq Q$, it follows that $e_0 \in K_1^*$, and hence by (iii'), $x = Ye_0$ is in K_2. Similarly, $x \in K_1$. □

We will see that, in general, $N(K_1, K_2)$ will be much smaller than $K_1 \cap K_2$.

The reason why we consider two convex cones instead of one is technical. We shall need only two special choices: either $K_1 = K_2 = K$ or $K_1 = K$, $K_2 = Q$. It is easy to see that

$$N(K_1 \cap K_2, K_1 \cap K_2) \subseteq N(K_1, K_2) \subseteq N(K_1 \cap K_2, Q).$$

This suggests that it would suffice to consider $N(K, K)$; but, as we shall see, $N(K, Q)$ behaves algorithmically better (see Theorem 1.6 and the remark following it), and this is why we allow two different cones. To simplify notation, we set $N(K) = N(K, Q)$ and $M(K) = M(K, Q)$. In this case, $K_2^* = Q^*$ is generated by the vectors e_i and f_i, and hence (iii') has the following convenient form:

(iii'') Every column of Y is in K; the difference of the first column and any other column is in K.

1.b. Properties of the cut operators. We give a lemma that yields a more explicit representation of constraints valid for $N(K)$ and $N_+(K)$. Unfortunately, the geometric meaning of $N(K)$ and $N_+(K)$ is not immediate; Lemmas 1.3 and 1.5 may be of some help in visualizing these constructions.

LEMMA 1.2. *Let $K \subseteq Q$ be a convex cone in $\mathbb{R}^{n+1}$ and $w \in \mathbb{R}^{n+1}$.*

(a) *$w \in N(K)^*$ if and only if there exist vectors $a_1, \cdots, a_n \in K^*$, a real number λ, and a skew symmetric matrix A such that $a_i + \lambda e_i + Ae_i \in K^*$ for $i = 1, \cdots, n$, and $w = \sum_{i=1}^n a_i + A\mathbf{1}$ (where $\mathbf{1}$ denotes the all-1 vector).*

(b) *$w \in N_+(K)^*$ if and only if there exist vectors $a_1, \cdots, a_n \in K^*$, a real number λ, a positive semidefinite symmetric matrix B, and a skew symmetric matrix A such that $a_i + \lambda e_i + Ae_i + Be_i \in K^*$ for $i = 1, \cdots, n$, and $w = \sum_{i=1}^n a_i + A\mathbf{1} + B\mathbf{1}$.*

Proof. Assume that $w \in N(K)^*$. Then $we_0^T \in M(K)^*$, and so we can write

$$we_0^T = \sum_i a_i b_i^T + \sum_{i=1}^n \lambda_i e_i f_i^T + A,$$

where $a_i \in K^*$, $b_i \in Q^*$, $\lambda_i \in \mathbb{R}$, and A is a skew symmetric matrix. Since Q^* is spanned by the vectors e_i and f_i, we may express the vectors b_i in terms of them and obtain a representation of the form

$$we_0^T = \sum_{i=1}^n a_i e_i^T + \sum_{i=1}^n \bar{a}_i f_i^T + \sum_{i=1}^n \lambda_i e_i f_i^T + A, \tag{1}$$

where a_i, $\bar{a}_i \in K^*$. Multiplying (1) by e_j from the right we get

$$0 = a_j - \bar{a}_j - \lambda_j e_j + Ae_j. \tag{2}$$

Multiplying (1) by e_0 and using (2) we get

$$w = \sum_{i=1}^{n} \bar{a}_i + \sum_{i=1}^{n} \lambda_i e_i + Ae_0 = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} Ae_i + Ae_0 = \sum_{i=1}^{n} a_i + A\mathbf{1}.$$

Here $a_j - \lambda_j e_j + Ae_j = \bar{a}_j \in K^*$. Since, trivially, $e_j \in K^*$, this condition remains valid if we decrease λ_j. Hence we can choose all the $\lambda_j = -\lambda$ equal. This proves the necessity of the condition given in (a).

The sufficiency of the condition, as well as of assertion (b), are proved by similar arguments. □

Our next lemma gives a geometric property of $N(K)$, which is easier to apply than the algebraic properties discussed before. Let $H_i = \{x \in \mathbb{R}^{n+1}: x_i = 0\}$ and $G_i = \{x \in \mathbb{R}^{n+1}: x_i = x_0\}$. Clearly, H_i and G_i are hyperplanes supporting Q at a facet, and all facets of Q are determined this way.

LEMMA 1.3. *For every convex cone $K \subseteq Q$ and every $1 \leqq i \leqq n$,*

$$N(K) \subseteq (K \cap H_i) + (K \cap G_i).$$

Proof. Consider any $x \in N(K)$ and let $Y \in M(K)$ be a matrix such that $Ye_0 = x$. Let y_i denote the ith column of Y. Then by (ii), $y_i \in G_i$ and by (iii″), $y_i \in K$, so $y_i \in K \cap G_i$. Similarly, $y_0 - y_i \in K \cap H_i$, and so $Ye_0 = y_0 = (y_0 - y_i) + y_i \in (K \cap H_i) + (K \cap G_i)$. □

Let us point out the following consequence of this lemma: if $K \cap G_i = \{0\}$, then $N(K) \subseteq H_i$. If, in particular, K meets both opposite facets of Q only in the 0 vector, then $N(K) = \{0\}$. This may be viewed as a very degenerate case of Gomory-Chvátal cuts (see below for more on the connection with Gomory-Chvátal cuts).

One could define a purely geometric cutting procedure based on this lemma: for each cone $K \subseteq Q$, consider the cone

$$N_0(K) = \cap_i ((K \cap G_i) + (K \cap H_i)). \tag{3}$$

This cone is similar to $N(K)$ but is generally bigger. We remark that this cone could also be obtained from a rather natural matrix cone by projection: this arises by imposing (ii), (iii), and the following restricted form of (i): $y_{0i} = y_{i0}$ for $i = 1, \cdots, n$.

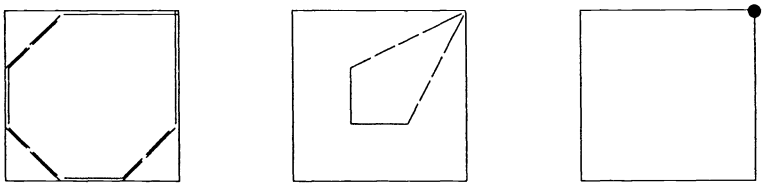
Figure 1 shows the intersection of three cones in $\mathbb{R}^3$ with the hyperplane $x_3 = 1$: the cones K, $N(K)$, and $N(N(K))$, and the constraints implied by Lemma 1.3. We see that the cone in Lemma 1.3 gets close to $N(K)$ but does not coincide with it.

We remark that $N(K \cap H_i) = N(K) \cap H_i$ for $i = 1, \cdots, n$; it should be noted that $N(K \cap H_i)$ does not depend on whether it is computed as a cone in $\mathbb{R}^{n+1}$ or in H_i.

We can get a better approximation of K^o by iterating the operator N. Define $N^t(K)$ recursively by $N^0(K) = K$ and $N^t(K) = N(N^{t-1}(K))$ for $t \geqq 1$.

THEOREM 1.4. $N^n(K) = K^o$.

Proof. Consider the unit cube Q' in the hyperplane $x_0 = 0$ and let $1 \leqq t \leqq n$. Consider any face F of Q' of dimension $n - t$ and let $\bar{F}$ be the union of faces of Q' parallel to



K $N(K)$ $N^2(K)$

FIG. 1

F. We prove, by induction on t, that

$$N^t(K) \subseteq \text{cone}\,(K \cap \bar{F}). \tag{4}$$

For $t = n$, this is just the statement of the theorem. For $t = 1$, this is equivalent to Lemma 1.3.

We may assume that F contains the vector e_0. Let F' be an $(n-t+1)$-dimensional face of Q' containing F and let i be an index such that $F' \cap H_i = F$. Then, by the induction hypothesis,

$$N^{t-1}(K) \subseteq \text{cone}\,(K \cap \overline{F'}).$$

Hence by Lemma 1.3,

$$N^t(K) = N(N^{t-1}(K)) \subseteq \text{cone}\,(N^{t-1}(K) \cap (H_i \cup G_i))$$
$$\subseteq \text{cone}\,([\text{cone}\,(K \cap \overline{F'}) \cap H_i] \cup [\text{cone}\,(K \cap \overline{F'}) \cap G_i]).$$

Now H_i is a supporting plane of cone $(K \cap \overline{F'})$ and hence its intersection with the cone is spanned by its intersection with the generating set of the cone:

$$\text{cone}\,(K \cap \overline{F'}) \cap H_i = \text{cone}\,(K \cap \overline{F'} \cap H_i) \subseteq \text{cone}\,(K \cap \bar{F}).$$

Similarly,

$$\text{cone}\,(K \cap \overline{F'}) \cap G_i \subseteq \text{cone}\,(K \cap \bar{F}).$$

Hence (4) follows. □

Next we show that if we use positive semidefiniteness, i.e., we consider $N_+(K)$, then an analogue of Lemma 1.3 can be obtained that is more complicated but important in the applications to combinatorial polyhedra.

LEMMA 1.5. *Let $K \subseteq Q$ be a convex cone and let $a \in \mathbb{R}^{n+1}$ be a vector such that $a_i \leqq 0$ for $i = 1, \cdots, n$ and $a_0 \geqq 0$. Assume that $a^T x \geqq 0$ is valid for $K \cap G_i$ for all i such that $a_i < 0$. Then $a^T x \geqq 0$ is valid for $N_+(K)$.*

(The condition that $a_0 \geqq 0$ excludes only trivial cases. The condition that $a_i \leqq 0$ is a normalization, which can be achieved by flipping coordinates.)

Proof. First, assume that $a_0 = 0$. Consider a subscript i such that $a_i < 0$. (If no such i exists, we have nothing to prove.) Then for every $x \in G_i \setminus \{0\}$, we have $a^T x \leqq a_i x_i < 0$, and so, $x \notin K$. Hence $K \cap G_i = \{0\}$, and so by Lemma 1.3, $N_+(K) \subseteq N(K) \subseteq K \cap H_i$. As this is true for all i with $a_i < 0$, we know that $a^T x = 0$ for all $x \in N_+(K)$.

Second, assume that $a_0 > 0$. Let $x \in N_+(K)$ and let $Y \in M_+(K)$ be a matrix with $Ye_0 = x$. For any $1 \leqq i \leqq n$, the vector Ye_i is in K by (iii″) and in G_i by (ii); so by the assumption on a, $a^T Ye_i \geqq 0$ whenever $a_i < 0$. Hence $a^T Y(a_0 e_0 - a) = a^T Y(-a_1 e_1 - \cdots - a_n e_n) \geqq 0$ (since those terms with $a_i = 0$ do not contribute to the sum anyway), and hence $a^T Y(a_0 e_0) \geqq a^T Ya \geqq 0$ by positive semidefiniteness. Thus $a^T Ye_0 = a^T x \geqq 0$. □

1.c. Algorithmic aspects. Next we turn to some algorithmic aspects of these constructions. We have to start by sketching the framework we are using; for a detailed discussion, see Grötschel, Lovász, and Schrijver [14].

Let K be a convex cone. A *strong separation oracle* for the cone K is a subroutine that, given a vector $x \in \mathbb{Q}^{n+1}$, either returns that $x \in K$ or returns a vector $w \in K^*$ such that $x^T w < 0$. A *weak separation oracle* is a version of this which allows for numerical errors: its input is a vector $x \in \mathbb{Q}^n$ and a rational number $\varepsilon > 0$, and it either returns the assertion that the euclidean distance of x from K is at most ε, or returns a vector w such that $|w| \geqq 1$, $w^T x \leqq \varepsilon$, and the euclidean distance of w from K^* is at most ε. If

the cone K is spanned by 0–1 vectors, then we can strengthen a weak separation oracle to a strong one in polynomial time.

Let us also recall the following consequence of the ellipsoid method: Given a weak separation oracle for a convex body, together with some technical information (say, the knowledge of a ball contained in the body and of another one containing the body), we can optimize any linear objective function over the body in polynomial time (again, allowing an arbitrarily small error). If we have a weak separation oracle for a convex cone $K \subseteq Q$, then we can consider its intersection with the halfspace $x_0 \leqq 1$; using the above result, we can solve various important algorithmic questions concerning K in polynomial time. We mention here the weak separation problem for the polar cone K^*.

THEOREM 1.6. *Suppose that we have a weak separation oracle for K. Then the weak separation problem for $N(K)$ as well as for $N_+(K)$ can be solved in polynomial time.*

Proof. Suppose that we have a (weak) separation oracle for the cone K. Then we have a polynomial time algorithm to solve the (weak) separation problem for the cone $M(K)$. In fact, let Y be any matrix. If it violates (i) or (ii), then this is trivially recognized and a separating hyperplane is also trivially given. (iii) can be checked as follows: we have to know if $Yu \in K$ holds for each $u \in Q^*$. Clearly it suffices to check this for the extreme rays of Q^*, i.e., for the vectors e_i and f_i. But this can be done using the separation oracle for K.

Since $N(K)$ is a projection of K, the weak separation problem for $N(K)$ can also be solved in polynomial time (by the general results from [14]).

In the case of $N_+(K)$, all we have to add is that the positive semidefiniteness of the matrix Y can be checked by Gaussian elimination, pivoting always on diagonal entries. If we always pivot positive elements, the matrix is positive semidefinite. If the test fails, it is easy to construct a vector v with $v^T Y v < 0$; this gives, then, a hyperplane separating Y from the cone. □

We remark that this proof does not remain valid for $N(K, K)$. In fact, let K be the cone induced by the incidence vectors of perfect matchings of a graph G with m nodes (with "1" appended as a 0th entry). Then the separation problem for K can be solved in polynomial time. On the other hand, consider the matrix $Y = (Y_{ij})$, where

$$y_{ij} = \begin{cases} 1, & \text{if } i = j \quad \text{or} \quad i = 0 \quad \text{or} \quad j = 0, \\ -4(m+2)/m^2, & \text{otherwise.} \end{cases}$$

Then $Y \in M(K, K)$ if and only if G is 3-edge-colorable, which is NP-complete to decide. We do not know if Theorem 1.6 extends to $N(K, K)$, but suspect that it does not.

Note, however, that if K is given by an explicit system of linear inequalities, then $M(K, K)$ is described by a system of linear inequalities of polynomial size and so the separation problem for $N(K, K)$ and $N_+(K, K)$ can be solved in polynomial time. In this case, we get a projection representation of $N(K)$ and of $N(K, K)$ from polyhedra with a polynomial number of facets. It should be remarked that this representation is canonical.

1.d. Stronger cut operators. We could use stronger versions of this procedure to get convex sets smaller than $N(K)$.

One possibility is to consider $N(K, K)$ instead of $N(K) = N(K, Q)$. It is clear that $N(K, K) \subseteq N(K)$. Trivially, Theorem 1.4 and Lemma 1.3 remain valid if we replace $N(K)$ by $N(K, K)$. Unfortunately, it is not clear whether Theorem 1.6 also remains valid. The problem is that now we have to check whether $YK^* \subseteq K$, and

unfortunately K^* may have exponentially many, or even infinitely many, extreme rays. If K is given by a system of linear inequalities, then this is not a problem. So in this case we could consider the sequence $N(K, K)$, $N(N(K, K), K)$, etc. This shrinks down faster to K^o than $N^t(K)$, as we shall see in the next section.

The following strengthening of the projection step in the construction seems quite interesting. For $v \in \mathbb{R}^{n+1}$, let $M(K)v = \{Yv: Y \in M(K)\}$. So $N(K) = M(K)e_0$. Now define

$$\hat{N}(K) = \bigcap_{v \in \operatorname{int}(Q^*)} M(K)v.$$

Note that the intersection can be written in the form

$$\hat{N}(K) = \bigcap_{u \in Q^*} M(K)(e_0 + u).$$

It is easy to see that

$$K^o \subseteq \hat{N}(K) \subseteq N(K).$$

The following lemma gives a different characterization of $\hat{N}(K)$.

LEMMA 1.7. *$x \in \hat{N}(K)$ if and only if for every $w \in \mathbb{R}^{n+1}$ and every $u \in Q^*$ such that $(e_0 + u)w^T \in M(K)^*$, we have $w^T x \geqq 0$.*

In other words, $\hat{N}(K)^*$ is generated by those vectors w for which there exists a $v \in \operatorname{int}(Q^*)$ such that $vw^T \in M(K)^*$.

Proof. (Necessity) Let $x \in \hat{N}(K)$, $w \in \mathbb{R}^{n+1}$, and $v \in \operatorname{int}(Q^*)$ such that $vw^T \in M(K)^*$. Then in particular x can be written as $x = Yv$, where $Y \in M(K)$. So $w^T x = w^T Yv = Y \cdot (vw^T) \geqq 0$.

(Sufficiency) Assume that $x \notin \hat{N}(K)$. Then there exists a $v \in \operatorname{int}(K^*)$ such that $x \notin M(K)v$. Now $M(K)v$ is a convex cone, and hence it can be separated from x by a hyperplane, i.e., there exists a vector $w \in \mathbb{R}^{n+1}$ such that $w^T x < 0$ but $w^T Yv \geqq 0$ for all $Y \in M(K)$. This latter condition means that $vw^T \in M(K)^*$, i.e., the condition given in the lemma is violated. □

The cone $\hat{N}(K)$ satisfies important constraints that the cones $N(K)$ and $N_+(K)$ do not. Let $b \in \mathbb{R}^{n+1}$, and define $F_b = \{x \in \mathbb{R}^{n+1}: b^T x \geqq 0\}$.

LEMMA 1.8. *Assume that $N(K \cap F_b) = \{0\}$. Then $-b \in \hat{N}(K)^*$.*

Proof. If $N(K \cap F_b) = \{0\}$, then for every matrix $Y \in M(K \cap F_b)$ we have $Ye_0 = 0$. In particular, $Y_{00} = 0$ and hence $Y = 0$. So $M(K \cap F_b) = \{0\}$. Since clearly

$$M(K \cap F_b)^* = M(K)^* + \operatorname{cone}\{bu^T: u \in Q^*\},$$

this implies that $M(K)^* + \{bu^T: u \in Q^*\} = \mathbb{R}^{(n+1)\times(n+1)}$. So, in particular, we can write $-be_0^T = Z + bu^T$ with $Z \in M(K)^*$ and $u \in Q^*$. Hence $-b(e_0 + u)^T \in M(K)^*$. By the previous lemma, this implies that $-b \in \hat{N}(K)^*$. □

We can use this lemma to derive a geometric condition on $\hat{N}(K)$ similar to Lemma 1.5.

LEMMA 1.9. *Let $K \subseteq Q$ be a convex cone and assume that $e_0 \notin K$. Then*

$$\hat{N}(K) \subseteq (K \cap G_1) + \cdots + (K \cap G_n).$$

In other words, if $a^T x \geqq 0$ is valid for all of the faces $K \cap G_i$, then it is also valid for $\hat{N}(K)$.

Proof. Let $b = -a + te_0$, where $t > 0$. Consider the cone $K \cap F_b$. By the definition of b, this cone does not meet any facet G_i of Q in any nonzero vector. Hence by Lemma 1.3, $N(K \cap F_b)$ is contained in every facet H_i of Q, and hence $N(K \cap F_b) \subseteq \operatorname{cone}(e_0)$. But $N(K \cap F_b) \subseteq K$ and so $N(K \cap F_b) = \{0\}$.

Hence by Lemma 1.7, we get that $-b = a - te_0 \in \hat{N}(K)^*$. Since this holds for every $t < \alpha$ and $\hat{N}(K)^*$ is closed, the lemma follows. □

Applying this lemma to the cone in Fig. 1, we can see that we obtain K^o in a single step. The next corollary of Lemma 1.9 implies that at least some of the Gomory–Chvátal cuts for K are satisfied by $\hat{N}(K)$.

COROLLARY 1.10. *Let $1 \leqq k \leqq n$ and assume that $\sum_{i=1}^{k} x_i > 0$ holds for every $x \in K$. Then $\sum_{i=1}^{k} x_i \geqq x_0$ holds for every $x \in \hat{N}(K)$.*

The proof consists of applying Lemma 1.9 to the projection of K on the first $k+1$ coordinates.

Unfortunately, we do not know if Theorem 1.6 remains valid for $\hat{N}(K)$. Of course, the same type of projection can be defined starting with $M_+(K)$ or with $M(K, K)$ instead of $M(K)$, and properties analogous to those in Lemmas 1.8, 1.9 can be derived.

2. Stable set polyhedra. We apply the results in the previous section to the stable set problem. To this end, we first survey some known methods and results on the facets of stable set polytopes.

2.a. Facets of stable set polyhedra and perfect graphs. Let $G = (V, E)$ be a graph with no isolated nodes. Let $\alpha(G)$ denote the maximum size of any stable set of nodes in G. For each $A \subseteq V$, let $\chi^A \in \mathbb{R}^V$ denote its incidence vector. The *stable set polytope* of G is defined as

$$\text{STAB}(G) = \text{conv}\{\chi^A : A \text{ is stable}\}.$$

So the vertices of STAB (G) are just the 0–1 solutions of the system of linear inequalities

$$x_i \geqq 0 \quad \text{for each } i \in V, \tag{1}$$

and

$$x_i + x_j \leqq 1 \quad \text{for each } ij \in E. \tag{2}$$

In general, STAB (G) is much smaller than the solution set of (1), (2), which we denote by FRAC (G) ("fractional stable sets"). In fact, they are equal if and only if the graph is bipartite. The polytope FRAC (G) has many nice properties; what we will need is that its vertices are half-integral vectors.

There are several classes of inequalities that are satisfied by STAB (G) but not necessarily by FRAC (G). Let us mention some of the most important classes. The *clique constraints* strengthen the class (2): for each clique B, we have

$$\sum_{i \in B} x_i \leqq 1. \tag{3}$$

Graphs for which (1) and (3) are sufficient to describe STAB (G) are called *perfect*. It was shown by Grötschel, Lovász, and Schrijver [12] that the weighted stable set problem can be solved in polynomial time for these graphs.

The *odd hole constraints* express the nonbipartiteness of the graph: if C induces a chordless odd cycle in G, then

$$\sum_{i \in C} x_i \leqq \tfrac{1}{2}(|C| - 1). \tag{4}$$

Of course, the same inequality holds if C has chords; but in this case it easily follows from other odd hole constraints and edge constraints. Nevertheless, it will be convenient that, if we apply an odd hole constraint, we do not have to check whether the circuit in question is chordless.

Graphs for which (1), (2), and (4) are sufficient to describe STAB (G) are called *t-perfect*. Graphs for which (1), (3), and (4) are sufficient are called *h-perfect*. It was shown by Grötschel, Lovász, and Schrijver [13] that the weighted stable set problem can be solved in polynomial time for h-perfect (and hence also for t-perfect) graphs.

The *odd antihole constraints* are defined by sets D that induce a chordless odd cycle in the complement of G:

$$\sum_{i \in D} x_i \leqq 2. \tag{5}$$

We shall see that the weighted stable set problem can be solved in polynomial time for all graphs for which (1)–(5) are enough to describe STAB (G) (and for many more graphs).

All constraints (2)–(5) are special cases of the *rank constraints*: let $U \subseteq V$ induce a subgraph G_U, then

$$\sum_{i \in U} x_i \leqq \alpha(G_U). \tag{6}$$

Of course, many of these constraints are inessential. To specify some that are essential, let us call a graph G *α-critical* if it has no isolated nodes and $\alpha(G-e) > \alpha(G)$ for every edge e. Chvátal [9] showed that if G is a connected α-critical graph then the rank constraint

$$\sum_{i \in V(G)} x_i \leqq \alpha(G)$$

defines a facet of STAB (G).

(Of course, in this generality, rank constraints are ill behaved: given any one of them, we have no polynomial time procedure to verify that it is indeed a rank constraint, since we have no polynomial time algorithm to compute the stability number of the graph on the right-hand side. For the special classes of rank constraints introduced above, however, it is easy to verify that a given inequality belongs to them.)

Finally, we remark that not all facets of the stable set polytope are determined by rank constraints. For example, let U induce an odd wheel in G, with center $u_0 \in U$. Then the constraint

$$\sum_{i \in U \setminus \{u_0\}} x_i + \frac{|U|-2}{2} x_{u_0} \leqq \frac{|U|-2}{2}$$

is called a *wheel constraint*. If, e.g., $V(G) = U$, then the wheel constraint induces a facet of the stable set polytope.

Another class of nonrank constraints of a rather different character are *orthogonality constraints*, introduced by Grötschel, Lovász, and Schrijver [12]. Let us associate with each vertex $i \in V$, a vector $v_i \in \mathbb{R}^n$, so that $|v_i| = 1$ and nonadjacent vertices correspond to orthogonal vectors. Let $c \in \mathbb{R}^n$ with $|c| = 1$. Then

$$\sum_{i \in V} (c^T v_i)^2 x_i \leqq 1$$

is valid for STAB (G). The solution set of these constraints (together with the nonnegativity constraints) is denoted by TH (G). It is easy to show that

$$\text{STAB}(G) \subseteq \text{TH}(G) \subseteq \text{FRAC}(G).$$

In fact, STAB (G) satisfies all the clique constraints. Note that there are infinitely many orthogonality constraints for a given graph, and TH (G) is in general nonpolyhedral (it is polyhedral if and only if the graph is perfect). The advantage of TH (G) is that every linear objective function can be optimized over it in polynomial time. The algorithm involves convex optimization in the space of matrices, and was the main motivation for our studies in the previous section. We shall see that these techniques

give substantially better approximations of STAB (G) over which one can still optimize in polynomial time.

2.b. The "N" operator. To apply the results in the previous chapter, we homogenize the problem by introducing a new variable x_0 and consider STAB (G) as a subset of the hyperplane H_0 defined by $x_0 = 1$. We denote by St (G) the cone spanned by the vectors

$$\begin{pmatrix} 1 \\ \chi^A \end{pmatrix} \in \mathbb{R}^{V \cup \{0\}},$$

where A is a stable set. We get STAB (G) by intersecting ST (G) with the hyperplane $x_0 = 1$. Similarly, let FR (G) denote the cone spanned by the vectors $\binom{1}{x}$, where $x \in$ FRAC (G). Then FR(G) is determined by the constraints

$$x_i \geqq 0 \quad \text{for each } i \in V,$$

and

$$x_i + x_j \leqq x_0 \quad \text{for each } ij \in E.$$

Since it is often easier to work in the original n-dimensional space (without homogenization), we shall use the notation $N(\text{FRAC}(G)) = N(\text{FR}(G)) \cap H_0$, and similarly for N_+, $\hat{N}$, etc. We shall also abbreviate $N(\text{FRAC}(G))$ by $N(G)$, etc. Since FRAC (G) is defined by an explicit linear program, one can solve the separation problem for it in polynomial time. We shall say briefly that the polytope is *polynomial time separable.* By Theorem 1.6, we obtain the following.

THEOREM 2.1. *For each fixed $r \geqq 0$, $N_+^r(G)$, as well as $N^r(G)$, are polynomial time separable.*

It should be remarked that, in most cases, if we use $N^r(G)$ as a relaxation of STAB (G), then it does not really matter whether the separation subroutine returns hyperplanes separating the given $x \notin N^r(G)$ from $N^r(G)$ or only from STAB (G). Hence it is seldom relevant to have a separation subroutine for a given relaxation, say, $N^r(G)$; one could use just as well a separation subroutine for any other convex body containing STAB (G) and contained in $N^r(G)$ (such as, e.g., $N_+^r(G)$). Hence the polynomial time separability of $N_+^r(G)$ is substantially deeper than the polynomial time separability of $N^r(G)$ (even though it does not imply it directly).

In the rest of this section we study the question of how much this theorem gives us: which graphs satisfy $N_+^r(G) = \text{STAB}(G)$ for small values of r, and more generally, which of the known constraints are satisfied by $N(G)$, $N_+(G)$, etc. With a little abuse of terminology, we shall not distinguish between the original and homogenized versions of clique, odd hole, etc., constraints.

It is a useful observation that if $Y = (y_{ij}) \in M(\text{FR}(G))$, then $y_{ij} = 0$ whenever $ij \in E(G)$. In fact, the constraint $x_i + x_j \leqq 1$ must be satisfied by Ye_i, and so $y_{ii} + y_{ji} \leqq y_{0i} = y_{ii}$ by nonnegativity. This implies $y_{ij} = 0$.

Let $a^T x \leqq b$ be any inequality valid for STAB (G). Let $W \subseteq V$ and let $a_W \in \mathbb{R}^W$ be the restriction of a to W. For every $v \in V$, if $a^T x \leqq b$ is valid for STAB (G), then $a_{V-v}^T x \leqq b$ is valid for STAB $(G - v)$ and $a_{V-\Gamma(v)-v}^T x \leqq b - a_v$ is valid for STAB $(G - \Gamma(v) - v)$. Let us say that these inequalities arise from $a^T x \leqq b$ by the *deletion* and *contraction* of node v, respectively. Note that if $a^T x \leqq b$ is an inequality such that for some v, both the deletion and contraction of v yield inequalities valid for the corresponding graphs, then $a^T x \leqq b$ is valid for G.

Let K be any convex body containing STAB (G) and contained in FRAC (G). Now Lemma 1.3 implies the following lemma.

LEMMA 2.2. *If $a^Tx \leqq b$ is an inequality such that for some $v \in V$, both the deletion and contraction of v give an inequality valid for K, then $a^Tx \leqq b$ is valid for $N(K)$.*

This lemma enables us to characterize completely the constraints obtained in one step (not using positive semidefiniteness).

THEOREM 2.3. *The polytope $N(G)$ is exactly the solution set of the nonnegativity, edge, and odd hole constraints.*

Proof. (1) It is obvious that $N(G)$ satisfies the nonnegativity and edge constraints. Consider an odd hole constraint $\sum_{i \in C} x_i \leqq \frac{1}{2}(|C|-1)$. Then for any $i \in C$, both the contraction and deletion of i result in an inequality trivially valid for FRAC (G). Hence the odd hole constraint is valid for $N(G)$ by Lemma 2.2.

(2) Conversely, assume that $x \in \mathbb{R}^V$ satisfies the nonnegativity, edge, and odd hole constraints. We want to show that there exists a nonnegative symmetric matrix $Y = (y_{ij}) \in \mathbb{R}^{(n+1)\times(n+1)}$ such that $y_{i0} = y_{ii} = x_i$ for all $1 \leqq i \leqq n$, $y_{00} = 1$, and

$$x_i + x_j + x_k - 1 \leqq y_{ik} + y_{jk} \leqq x_k$$

for all $i, j, k \in V$ such that $ij \in E$ (the lower bound comes from the condition that $Yf_k \in \mathrm{FR}(G)$; the upper, from the condition that $Ye_k \in \mathrm{FR}(G)$). Note that the constraint has to hold in particular when $i = k$; then the upper bound implies that $y_{ij} = 0$, while the lower bound is automatically satisfied.

The constraints on the y's are of a special form: they involve only two variables. We can therefore use the following (folklore) lemma, which gives a criterion for the solvability of such a system, more combinatorial than the Farkas lemma.

LEMMA 2.4. *Let $H = (W, F)$ be a graph and let two values $0 \leqq a(ij) \leqq b(ij)$ be associated with each edge of H. Let $U \subseteq W$ also be given. Then the linear system*

$$a(ij) \leqq y_i + y_j \leqq b(ij) \qquad (ij \in F),$$

$$y_i \geqq 0 \qquad (i \in W),$$

$$y_i = 0 \qquad (i \in U)$$

has no solution if and only if there exists a sequence of (not necessarily distinct) vertices $v_0, v_1, \cdots, v_p$ such that v_i and v_{i+1} are adjacent (the sequence is a walk), and one of the following holds:

(a) *p is odd and $b(v_0v_1) - a(v_1v_2) + b(v_2v_3) - \cdots + b(v_{p-1}v_p) < 0$;*

(b) *p is even, $v_0 = v_p$, and $b(v_0v_1) - a(v_1v_2) + b(v_2v_3) - \cdots - a(v_{p-1}v_p) < 0$;*

(c) *p is even, $v_p \in U$, and $b(v_0v_1) - a(v_1v_2) + b(v_2v_3) - \cdots - a(v_{p-1}v_p) < 0$;*

(d) *p is odd, $v_0, v_p \in U$, and $-a(v_0v_1) + b(v_1v_2) - a(v_2v_3) - \cdots - a(v_{p-1}v_p) < 0$.*

In our case, we have as W the set of all pairs $\{i, j\}$ $(i \neq j)$, U is the subset consisting of the edges of G, two pairs, $\{i, j\}$ and $\{k, l\}$, are adjacent in H if and only if $i = k$ and $je \in E(G)$, and $a(ij, jk) = x_i + x_j + x_k - 1$, $b(ij, jk) = x_j$. We want to verify that if x satisfies all the odd hole constraints, then none of the walks of types (a)-(d) in the lemma above can occur. Let us ignore, for a while, how the walk ends. The vertices of the walk in H correspond to pairs ij; the edges in the walk correspond to triples (ijk) such that $ik \in E$. Let us call this edge the *bracing edge* of the triple. We have to add up alternately x_j and $1 - x_i - x_j - x_k$; call the triple *positive* and *negative* accordingly.

Let w be a vertex of G that is not an element of the first and last pair v_0 and v_p. Then following the walk, w may become an element of a v_i, stay an element for a while, and then cease to be; this may be repeated, say, $f(w)$ times. It is then easy to see that the total contribution of the variable x_w to the sum is $-f(w)x_w$.

It is easy to settle case (b) now. Then any v_i can be considered first, and so the above counting applies to each vertex (unless all pairs v_i share a vertex of G, which is a trivial case). So the sum

$$b(v_0v_1)-a(v_1v_2)+b(v_2v_3)-\cdots-a(v_{p-1}v_p)=\frac{p}{2}-\sum_w f(w)x_w.$$

But note that every vertex w occurs in exactly $2f(w)$ bracing edges. If we add up the edge constraints for all bracing edges, we get $p-\sum_w 2f(w)x_w \geqq 0$, which shows that (b) cannot occur.

Cases (a) and (c) take only a little care around the end of the walk, and are left to the reader. Let us show how case (d) can be settled, which is the only case in which the odd hole constraints are needed.

Consider again the bracing edges of the triples, but now, count the pairs v_0 and v_p (which are edges of G) as bracing edges. Again, it is easy to see that the total sum in question is $(p+1)/2-\sum f(w)x_w$, where each w is contained in exactly $2f(w)$ bracing edges. Unfortunately, we now have $p+2$ bracing edges, so adding up the edge constraints for them would not yield the nonnegativity of the sum. But observe that the multiset of bracing edges (we count an edge that is bracing in more than one triple with multiplicity) forms an Eulerian graph, and is, therefore, the union of circuits. Since the total number of bracing edges, $p+2$, is odd, at least one of these circuits is odd. Add up the odd hole constraint for this circuit and the edge constraint, divided by two, for each of the remaining bracing edges. We get that $\sum_w f(w)x_w \leqq (p+1)/2$, which shows that (d) cannot occur. □

COROLLARY 2.5. *If G is t-perfect, then* STAB(G) *is the projection of a polytope whose number of facets is polynomial in n. Moreover, this representation is canonical.* □

This corollary generalizes a result of Barahona and Mahjoub [5] that constructs a projection representation for series-parallel graphs. It could also be derived in an alternative way. The separation problem for the odd cycle inequalities can be reduced to n shortest path problems (see [13]). Following this construction, one can see that a vector x is in the stable set polytope of a t-perfect graph if and only if n potential functions exist in an auxiliary graph. This yields a representation of STAB(G) as the projection of a polytope with $O(n^2)$ facets. (We are grateful to the referee for this remark.)

2.c. The repeated "N" operator. Next, we prove a theorem which describes a large class of inequalities valid for $N^r(G)$ for a given r. The result is not as complete as in the case $r=1$, but it does show that the number of constraints obtainable grows very quickly with r.

Let $a^Tx \leqq b$ be any inequality valid for STAB(G). By Theorem 1.4, there exists an $r \geqq 0$ such that $a^Tx \leqq b$ is valid for $N^r(G)$. Let the *N-index* of the inequality be defined as the least r for which this is true. We can define (and will study later) the N_+-index analogously. Note that in each version, the index of an inequality depends only on the subgraph induced by those nodes having a nonzero coefficient. In particular, if these nodes induce a bipartite graph, then the inequality has N-index 0. We can define the N-index of a graph as the largest N-index of the facets of STAB(G). The N-index of G is 0 if and only if G is bipartite; the N-index of G is 1 if and only if G is t-perfect. Lemma 2.2 implies the following corollary (using the obvious fact that the N-index of an induced subgraph is never larger than the N-index of the whole graph).

COROLLARY 2.6. *If for some node v, $G-v$ has N-index k, then G has N-index at most $k+1$.* □

The following lemma about the iteration of the operator N will be useful in estimating the N-index of a constraint.

LEMMA 2.7. $1/(k+2)\mathbf{1} \in N^k(G)$ $(k \geqq 0)$.

Proof. We use induction on k. The case $k=0$ is trivial. Consider the matrix $Y=(y_{ij}) \in \mathbb{R}^{(V\cup\{0\})\times(V\cup\{0\})}$ defined by

$$y_{ij}=\begin{cases}1 & \text{if } i=j=0,\\ 1/(k+1), & \text{if } i=0 \text{ and } j>0 \text{ or } i>0 \text{ and } j=0 \text{ or } i=j>0,\\ 0, & \text{otherwise.}\end{cases}$$

Then $Y \in M(N^{k-1}(\mathrm{FR}(G)))$, since

$$Ye_i=\frac{1}{k+2}(e_0+e_i) \in \mathrm{ST}(G) \subseteq N^{k-1}(\mathrm{FR}(G))$$

and

$$Yf_i=\frac{k+1}{k+2}e_0+\sum_{j\neq 0,i}\frac{1}{k+2}e_j \leqq \frac{k+1}{k+2}\left(e_0+\frac{1}{k+1}\sum_{j\in V}e_j\right) \in N^{k-1}(\mathrm{FR}(G)),$$

and so by the monotonicity of $N^{k-1}(\mathrm{FR}(G))$, $Yf_i \in N^{k-1}(\mathrm{FR}(G))$. Hence the first column of Y is in $N^k(\mathrm{FR}(G))$, and thus $1/(k+2)\mathbf{1} \in N^k(G)$. □

From these two facts, we can derive some useful bounds on the N-index of a graph.

COROLLARY 2.8. *Let G be a graph with n nodes and at least one edge. Assume that G has stability number $\alpha(G)=\alpha$ and N-index k. Then*

$$\frac{n}{\alpha}-2 \leqq k \leqq n-\alpha-1.$$

Proof. The upper bound follows from Corollary 2.6, applying it repeatedly to all but one nodes outside a maximum stable set. To show the lower bound, assume that $k<(n/\alpha)-2$. Then the vector $(1/(k+2))\mathbf{1}$ does not satisfy the constraint $\sum_i x_i \leqq \alpha$ and so it does not belong to STAB (G). Since it belongs to $N^k(G)$ by Lemma 2.7, it follows that $N^k(G) \neq \mathrm{STAB}(G)$—a contradiction. □

It follows in particular that the N-index of a complete graph on t vertices is $t-2$. The N-index of an odd hole is 1, as an odd whole is a t-perfect graph. The N-index of an odd antihole with $2k+1$ nodes is k; more generally, we have the following corollary.

COROLLARY 2.9. *The N-index of a perfect graph G is $w(G)-2$. The N-index of a critically imperfect graph G is $w(G)-1$.*

Next we study the index of a single inequality. Let $a^Tx \leqq b$ be any constraint valid for STAB (G) $(a \in \mathbb{Z}_+^V, b \in \mathbb{Z}_+)$. Define the *defect* of this inequality as $2\times \max\{a^T-b: x \in \mathrm{FRAC}(G)\}$. The factor 2 in front guarantees that this is an integer. In the special case when we consider the constraint $\sum_i x_i \leqq \alpha(G)$ for an α-critical graph G, the defect is just the *Gallai class number* of the graph (see Lovász and Plummer [18] for a discussion of α-critical graphs, in particular of the Gallai class number).

Given a constraint, its defect can be computed in polynomial time, since optimizing over FRAC (G) is an explicit linear program. The defect of a constraint is particularly easy to compute if the constraint defines a facet of STAB (G). This is shown by the following lemma, which states a property of facets of STAB (G) of independent interest.

LEMMA 2.10. *Let $\sum_i a_i x_i \leqq b$ define a facet of* STAB (G), *different from those determined by the nonnegativity and edge constraints. Then every vector v maximizing*

$a^T x$ over FRAC (G) *has* $v_i = \frac{1}{2}$ *whenever* $a_i > 0$. *In particular,*

$$\max\{a^T x: x \in \text{FRAC}(G)\} = \tfrac{1}{2}\sum_i a_i$$

and the defect of the inequality is $\sum_i a_i - 2b$.

Proof. Let v be any vertex of FRAC (G) maximizing $a^T x$. It suffices to prove that $v_i \neq 1$ whenever $a_i > 0$; this will imply that the vector $(\frac{1}{2}, \cdots, \frac{1}{2})^T$ also maximizes $a^T x$, and to achieve the same objective value, v must have $v_i = \frac{1}{2}$ whenever $a_i > 0$.

Let $U = \{i \in V: v_i = 1\}$ and assume, by way of contradiction, that $a(U) > 0$. Clearly U is a stable set. If we choose v so that U is minimal (but of course nonempty), then $a_i > 0$ for every $i \in U$. Let $\Gamma(U)$ denote the set of neighbors of U. Let X be any stable set in G whose incidence vector χ^X is a vertex on the facet of STAB (G) determined by $a^T x = b$.

Consider the set $Y = U \cup (X \setminus \Gamma(U))$. Clearly, Y is stable and $a(Y) = a(X) + a(U \setminus X) - a(\Gamma(U) \cap X)$. So, by the optimality of X, we have

$$a(U \setminus X) \leqq a(\Gamma(U) \cap X).$$

On the other hand, consider the vector $w \in \mathbb{R}^V$ defined by

$$w_i = \begin{cases} 1, & \text{if } i \in U \cap X, \\ 0, & \text{if } i \in \Gamma(U) \setminus X, \\ \frac{1}{2}, & \text{otherwise.} \end{cases}$$

Then $w \in$ FRAC (G) and $a^T w \geqq a^T v + \frac{1}{2}a(\Gamma(U) \cap X) - \frac{1}{2}a(U \setminus X) \geqq a^T v$. By the optimality of v, we must have equality, and so $a(U \setminus X) = a(\Gamma(U) \cap X)$. But this means that χ^X satisfies the linear equation

$$\sum_{i \in U \cup \Gamma(U)} a_i x_i = a(U).$$

So this linear equation is satisfied by every vertex of the facet determined by $a^T x = b$. The only way this can happen is that it is the equation $a^T x = b$ itself. But then $a^T v = b$ and so $a^T v \leqq b$ also defines a facet of FRAC (G), which was excluded. □

We need some further, related lemmas about stable set polytopes. These may be viewed as weighted versions of results on graphs with the so-called König property; see [18, § 6.3].

LEMMA 2.11. *Let* $a \in \mathbb{R}_+^V$ *and assume that*

$$\max\{a^T x: x \in \text{STAB}(G)\} < \max\{a^T x: x \in \text{FRAC}(G)\}.$$

Let E' *be the set of those edges* ij *for which* $y_i + y_j = 1$ *holds for every vector* $y \in$ FRAC (G) *maximizing* $a^T x$. *Then* (V, E') *is nonbipartite.*

Proof. Suppose that (V, E') is bipartite. Let z be a vector in the relative interior of the face F of FRAC (G) maximizing $a^T x$. Then clearly

$$E' = \{ij \in E: z_i + z_j = 1\}$$

and

$$F = \{x \in \text{FRAC}(G): x_i + x_j = 1 \text{ for all } ij \in E\}.$$

Let (U, W) be a bipartition of (V, E'). In every connected component of (V, E'), $z_i \geqq \frac{1}{2}$ on at least one color class and hence we may choose (U, W) so that $z_i \geqq \frac{1}{2}$ for all $i \in W$. Then, W is a stable set in the whole graph G. Hence it follows that $\chi^W \in F$. This implies that $\max\{a^T x: x \in \text{STAB}(G)\} = \max\{a^T x: x \in \text{FRAC}(G)\}$—a contradiction. □

LEMMA 2.12. *As in the previous lemma, let* $a \in \mathbb{R}_+^V$ *and assume that*

$$\max\{a^T x: x \in \mathrm{STAB}(G)\} < \max\{a^T x: x \in \mathrm{FRAC}(G)\}.$$

Then there exists an $i \in V$ *such that every vector* $y \in \mathrm{FRAC}(G)$ *maximizing* $a^T x$ *has* $y_i = \frac{1}{2}$.

Proof. Let E' be as before. Then by Lemma 2.11, there exists an odd circuit C in G such that $E(C) \subseteq E'$. If y is any vector in FRAC (G) maximizing $a^T x$, then by the definition of E', $y_i + y_j = 1$ for every edge $ij \in E(C)$, and hence $y_i = \frac{1}{2}$ for every $i \in V(C)$. □

Now we can state and prove our theorem, which shows the connection between defect and the N-index.

THEOREM 2.13. *Let* $a^T x \leqq b$ *be an inequality with integer coefficients valid for* STAB (G) *with defect* r *and* N*-index* k. *Then*

$$\frac{r}{b} \leqq k \leqq r.$$

Proof. (Upper bound) We use induction on r. If $r = 0$ we have nothing to prove, so suppose that $r > 0$. Then Lemma 2.12 can be applied and we get that there is a vertex i such that every vector y optimizing $a^T x$ over FRAC (G) has $y_i = \frac{1}{2}$. Note that trivially $a_i > 0$.

We claim that both the contraction and deletion of i result in constraints with smaller defect. In fact, let y be a vertex of FRAC (G) maximizing $a_{V-i}^T x$. If y also maximizes $a^T x$, then $y_i = \frac{1}{2}$ and hence

$$2(a_{V-i}^T y - b) = 2(a^T y - b) - a_i < 2(a^T y - b) = r.$$

On the other hand, if y does not maximize $a^T x$, then

$$2(a_{V-i}^T y - b) \leqq 2(a^T y - b) < 2 \cdot \max\{a^T x - b: x \in \mathrm{FRAC}(G)\} = r.$$

The assertion follows similarly for the contraction. Hence by the induction hypothesis, the contraction and deletion of i yield constraints valid for $N^{r-1}(G)$. It follows by Lemma 2.2 that $a^T x \leqq b$ is valid for $N^r(G)$.

(Lower bound) By Lemma 2.7, $(1/(k+2))\mathbf{1} \in N^k(G)$, and so $a^T x \leqq b$ must be valid for $(1/(k+2))\mathbf{1}$. So $(1/(k+2))a^T\mathbf{1} \leqq b$ and hence

$$k \geqq \frac{a^T\mathbf{1}}{b} - 2 = \frac{r}{b}.$$ □

It follows from our discussions that for an odd antihole constraint, the lower bound is tight. On the other hand, it is not difficult to check that for a rank constraint defined by an α-critical subgraph that arises from K_p by subdividing an edge by an even number of nodes, the upper bound is tight.

We would like to mention that Ceria [7] proved that $N(\mathrm{FRAC}(G), \mathrm{FRAC}(G))$ also satisfies, among others, the K_4-constraints. We do not study the operator $K \mapsto N(K, K)$ here in detail, but a thorough comparison of its strength with N and N_+ would be very interesting.

A class of graphs interesting from the point of view of stable sets is the class of line-graphs: the stable set problem for these graphs is equivalent to the matching problem. In particular, it is polynomial time solvable and Edmonds's description of the matching polytope [10] provides a "nice" system of linear inequalities describing the stable set polytope of such graphs. The N-index of line-graphs is unbounded; this follows, e.g., by Corollary 2.8. This also follows from Yannakakis's result [26] mentioned in the Introduction, since bounded N-index would yield a representation of the matching polytope as a projection of a polytope with a polynomial number of facets. We do not know whether or not the N_+-index of line-graphs remains bounded.

2.d. The "N_+" operator. Now we turn to the study of the operator N_+ for stable set polytopes. We do not have as general results for the operator N_+ as for the operator N, but we will be able to show that many constraints are satisfied even for very small r.

Lemma 1.5 implies the following lemma.

LEMMA 2.14. *If $a^Tx \leqq b$ is an inequality valid for* STAB (G) *such that for all $v \in V$ with a positive coefficient the contraction of v gives an inequality with N_+-index at most r, then $a^Tx \leqq b$ has N_+-index at most $r+1$.*

The clique, odd hole, odd wheel, and odd antihole constraints have the property that, contracting any node with a positive coefficient, we get an inequality in which the nodes with positive coefficients induce a bipartite subgraph. Hence, we have the following corollary.

COROLLARY 2.15. *Clique, odd hole, odd wheel, and odd antihole constraints have N_+-index* 1.

Hence all h-perfect (in particular all perfect and t-perfect) graphs have N_+-index at most 1. We can also formulate the following recursive upper bound on the N_+-index of a graph.

COROLLARY 2.16. *If $G-\Gamma(v)-v$ has N_+-index at most r for every $v \in V$, then G has N_+-index at most $r+1$.*

Next, we consider the orthogonality constraints. To this end, consider the cone M_{TH} of $(V \cup \{0\}) \times (V \cup \{0\})$ matrices $Y = (y_{ij})$ satisfying the following constraints:

(i) Y is symmetric;

(ii) $y_{ii} = y_{i0}$ for every $i \in V$;

(iii′) $y_{ij} = 0$ for every $ij \in E$;

(iv) Y is positive semidefinite.

As remarked, (iii′) is a relaxation of (iii) in the definition of $M_+(\mathrm{FR}(G))$. Hence $M_+(\mathrm{FR}(G)) \subseteq M_{\mathrm{TH}}$.

LEMMA 2.17. $\mathrm{TH}(G) = \{Ye_0 \colon Y \in M_{\mathrm{TH}}, e_0^T Ye_0 = 1\}$.

Proof. Let $x \in \mathrm{TH}(G)$. Then, by the results of Grötschel, Lovász, and Schrijver [13], x can be written in the form $x_i = (v_0^T v_i)^2$, where the v_i $(i \in V)$ form an orthonormal representation of the complement of G and v_0 is some vector of unit length. Set $x_0 = 1$ and define $Y_{ij} = v_i^T v_j \sqrt{x_i x_j}$. Then it is easy to verify that $Y \in M_{\mathrm{TH}}$ and $Ye_0 = x$.

The converse inclusion follows by a similar direct construction. □

This representation of $\mathrm{TH}(G)$ is not a special case of the matrix cuts introduced in § 1 (though it is clearly related). In § 3 we will see that, in fact, $\mathrm{TH}(G)$ is in a sense more fundamental than the relaxations of $\mathrm{STAB}(G)$ constructed in § 1. Right now we can infer the following.

COROLLARY 2.18. *Orthogonality constraints have N_+-index* 1.

We conclude with an upper bound on the N_+-index of a single inequality. Since $\alpha(G-\Gamma(v)-v) < \alpha(G)$, Lemma 2.14 gives, by induction, Corollary 2.19.

COROLLARY 2.19. *If $a^Tx \leqq b$ is an inequality valid for* STAB (G) *such that the nodes with positive coefficient induce a graph with independence number r, then $a^Tx \leqq b$ has N_+-index at most r. In particular, $a^Tx \leqq b$ has index at most b.*

Let us turn to the algorithm aspects of these results. Theorem 2.1 implies the following corollary.

COROLLARY 2.20. *The maximum weight stable set problem is polynomial time solvable for graphs with bounded N_+-index.*

Note that even for small values of r, quite a few graphs have N_+-index at most r. Collecting previous results, we obtain Corollary 2.21.

COROLLARY 2.21. *For any fixed $r \geqq 0$, if* STAB (G) *can be defined by constraints $a^Tx \leqq b$ such that either the defect of the constraint is at most r or the support contains*

no stable set larger than r, then the maximum weight stable set problem is polynomial time solvable for G.

3. Cones of set-functions. Vectors in $\mathbb{R}^S$ are just functions defined on the one-element subsets of a set S; the symmetric matrices in the previous sections can be considered as functions defined on unordered pairs. We show that if we consider set-functions, i.e., functions defined on all subsets of S, then some of the previous considerations become more general and sometimes even simpler.

In fact, most of the results extend to a general finite lattice in the place of the boolean algebra, and we present them in this generality for the sake of possible other applications.

3.a. Preliminaries: Vectors on lattices. Let us start with some general facts about functions defined on lattices. Given a lattice L, we associate with it the matrix $Z = (\zeta_{ij})$, called the *zeta-matrix* of the lattice, defined by

$$\zeta_{ij} = \begin{cases} 1, & \text{if } i \leqq j, \\ 0, & \text{otherwise.} \end{cases}$$

For $j \in L$, let ζ^j denote the jth column of the zeta matrix, i.e., let

$$\zeta^j(i) = \zeta_{ij}.$$

If we order the rows and columns of Z compatibly with the partial ordering defined by the lattice, it will be upper triangular with 1's in its main diagonal. Hence it is invertible, and its inverse $M = Z^{-1}$ is an integral matrix of the same shape. This inverse is a very important matrix, called the *Möbius matrix* of the lattice. Let

$$M = (\mu(i, j))_{i,j \in \mathscr{L}}.$$

The function μ is called the *Möbius function* of the lattice. From the discussion above, we see that $\mu(i, i) = 1$ for all $i \in \mathscr{L}$, and $\mu(i, j) = 0$ for all $i, j \in \mathscr{L}$ such that $i \not\leqq j$. Moreover, the definition of M implies that for every pair of elements $a \leqq b$ of the lattice,

$$\sum_{a \leqq i \leqq b} \mu(a, i) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{otherwise;} \end{cases}$$

and

$$\sum_{a \leqq i \leqq b} \mu(i, b) = \begin{cases} 1, & \text{if } a = b, \\ 0, & \text{otherwise.} \end{cases}$$

Either one of these identities provides a recursive procedure to compute the Möbius function. It is easy to see from this procedure that the value of the Möbius function $\mu(i, j)$, where $i \leqq j$, depends only on the internal structure of the interval $[i, j]$. Also note the symmetry in these two identities. This implies that if μ^* denotes the Möbius function of the lattice turned upside down, then

$$\mu^*(i, j) = \mu(j, i).$$

For $j \in L$, let μ^j denote the jth column of the Möbius matrix, i.e., let

$$\mu^j(i) = \mu_{ij}.$$

We denote by μ_j the jth row of the Möbius matrix, and by $\mu_{[i,j]}$ the restriction of μ_i to the interval $[i, j]$, i.e., the vector defined by

$$\mu_{[i,j]}(k) = \begin{cases} \mu(i, k), & \text{if } k \leqq j, \\ 0, & \text{otherwise.} \end{cases}$$

The Möbius function of a lattice generalizes the Möbius function in number theory, and it can be used to formulate an inversion formula extending the Möbius inversion in number theory. Let $g \in \mathbb{R}^L$ be a function defined on the lattice. The zeta matrix can be used to express its *lower* and *upper summation function*:

$$(Z^T g)(i) = \sum_{j \leqq i} g(j),$$

and

$$(Zg)(i) = \sum_{j \geqq i} g(j).$$

Given (say) $f = Zg$, we can recover g uniquely by

$$g(i) = (Mf)(i) = \sum_{j \geqq i} \mu(i, j) f(j).$$

The function g is called the *upper Möbius inverse* of f. The *lower Möbius inverse* is defined analogously.

There is a further simple but important formula relating a function to its inverse. Given a function $f \in \mathbb{R}^L$, we associate with it the matrix $W^f = (w_{ij})$, where

$$w_{ij} = f(i \vee j).$$

We also consider the diagonal matrix D^f with $(D^f)_{ii} = f(i)$. Then it is not difficult to prove the following identity (Lindström [15], Wilf [24]).

LEMMA 3.1. *If g is the upper Möbius inverse of f, then $W^f = ZD^g Z^T$.*

For more on Möbius functions, see Rota [21], Lovász [17, Chap. 2], or Stanley [23, Chap. 3].

A function $f \in \mathbb{R}^L$ will be called *strongly decreasing* if $Mf \geqq 0$. Since $f = Z(Mf)$, this is equivalent to saying that f is a nonnegative linear combination of the columns of Z, i.e., of the vectors ζ_j. So strongly decreasing functions form a convex cone $H = H(L)$, which is generated by the vectors ζ^j, $j \in L$. Also by definition, the polar cone H^* is generated by the rows of M, i.e., by the vectors μ_j.

Let us mention that the vector $\mu_{[i,j]}$ is also in H^* for every $i \leqq j$. This is straightforward to check by calculating the inner product of $\mu_{[i,j]}$ with the generators ζ_j of H. It is easy to see that strongly decreasing functions are nonnegative, monotone decreasing, and *supermodular*, i.e., they satisfy

$$f(i \vee j) + f(i \wedge j) \geqq f(i) + f(j).$$

Lemma 3.1 implies Corollary 3.2.

COROLLARY 3.2. *A function f is strongly decreasing if and only if W^f is positive semidefinite.*

It follows, in particular, that f is strongly decreasing if and only if for every $x \in \mathbb{R}^L$,

$$x^T W^f x = \sum_{x_i x_j} f(i \vee j) \geqq 0.$$

It is, in fact, worthwhile to mention the following identity, following immediately from Lemma 3.1. Let f, $x \in \mathbb{R}^L$ and let $g = Mf$ and $y = Zx$. Then

$$x^T W^f x = \sum_{i \in L} g(i) y(i)^2.$$

In particular, if f is strongly decreasing, then

$$x^T W^f x \geqq g(0) x(0)^2. \tag{1}$$

Remark. Let $L = 2^S$, and let $f \in \mathbb{R}^L$ such that $f(\emptyset) = 1$. Then f is strongly decreasing if and only if there exist random events A_s $(s \in S)$ such that for every $X \subseteq S$,

$$\text{Prob}\left(\prod_{s \in X} A_s\right) = f(X).$$

(If this is the case, $(Mf)(X)$ is the probability of the atom $\prod_{s \in X} A_s \prod_{s \in S - X} \bar{A}_s$.) In particular, we obtain from (1) that for any $\lambda \in \mathbb{R}^L$ with $\lambda(0) = 1$,

$$\sum_{X,Y} \lambda_X \lambda_Y \text{ Prob}\left(\prod_{s \in X \cup Y} A_s\right) \geqq \text{Prob}\left(\prod_{i \in S} \bar{A}_i\right).$$

This is a combinatorial version of the Selberg sieve in number theory (see [17, Chap. 2]). Inequality (1) can be viewed as Selberg's sieve for general lattices; see Wilson [25].

The lattice structure also induces a "multiplication," which leads to the *semigroup algebra* of the semigroup $(L, \vee)$. Given $a, b \in \mathbb{R}^L$, we define the vector $a \vee b \in \mathbb{R}^L$ by

$$(a \vee b)(k) = \sum_{i \vee j = k} a(i) b(j).$$

In particular,

$$e_i \vee e_j = e_{i \vee j}$$

(and the rest of the definition is obtained by distributivity). It is straightforward to see that this operation is commutative, associative, and distributive with respect to the vector addition, and has unit element e_0 (where 0 is the zero element of the lattice). This semigroup algebra has a very simple structure: elementary calculations show that

$$Z^T(a \vee b)(k) = (Z^T a)(k) \cdot (Z^T b)(k), \tag{2}$$

and hence the semigroup algebra is isomorphic to the direct product of $|L|$ copies of $\mathbb{R}$. It also follows from (2) that a vector a has an inverse in this algebra if and only if $(Z^T a)(k) \neq 0$ for all k.

Another identity which will be useful is the following:

$$(a \vee b)^T c = a^T W^c b. \tag{3}$$

Using this, we can express the fact that a vector c is strongly decreasing as follows:

$$(a \vee a)^T c \geqq 0 \quad \text{for every } a \in \mathbb{R}^L.$$

In particular it follows that H^* *is generated by the vectors* $a \vee a$, $a \in \mathbb{R}^L$. Comparing this with our previous characterization, it follows that the vectors μ_j must be of the form $a \vee a$. In fact, $\mu_j \vee \mu_j = \mu_j$; more generally, the vectors $\mu_{[i,j]}$ are also idempotent. Using (2) it is easy to see that the idempotents are exactly the vectors of the form $\sum_{i \in I} \mu_i$, where $I \subseteq L$. Moreover, the "$\vee$" product of any two vectors μ_i is zero.

3.b. Optimization in lattices. Given a subset $F \subseteq L$, we denote by cone (F) the convex cone spanned by the vectors ζ^i, $i \in F$. Since these vectors are extreme rays of H, and all extreme rays of H are linearly independent, it is, in principle, trivial to describe F by linear inequalities. It is determined by the system

$$\mu_j^T x \begin{cases} = 0, & \text{if } i \notin F, \\ \geqq 0, & \text{if } i \in F. \end{cases} \tag{4}$$

But since cone (F) is generally not full-dimensional, it may have many other minimal descriptions. For example, in the case when F is an order ideal (i.e., $x \in F$, $y \leqq x$ imply $y \in F$), cone (F) could be described by

$$x \in H, \qquad x(i) = 0 \quad \text{for all } i \notin F. \tag{5}$$

Hence

$$\text{cone}\,(F)^* = \{a \in \mathbb{R}^L : (Z^T a)(k) \geqq 0 \text{ for all } k \in F\}. \tag{6}$$

Our main concern will be to describe the projection of cone (F) on the subspace spanned by a few "small" elements in the lattice. Let I be the set of these "interesting" lattice elements. We consider $\mathbb{R}^I$ as the subspace of $\mathbb{R}^L$ spanned by the elements of I. For any convex cone $k \subseteq H$, let K_I denote the intersection of K with $\mathbb{R}^I$ and let K/I denote the projection of K onto $\mathbb{R}^I$. Then $(K^*)_I \subseteq K^*$ is the set of linear inequalities valid for K involving only variables corresponding to elements of I. Also, $(K^*)_I$ is the polar of K/I with respect to the linear space $\mathbb{R}^I$.

For example, in the case when $L = 2^S$, where S is an n-element set, we can take I as the set of all singletons and $\emptyset$. If we project cone (F) on this subspace, and intersect the projection with the hyperplane $x_\emptyset = 1$, then we recover the polyhedron usually associated with F (namely, the convex hull of incidence vectors of members of F). Note that the projection itself is just the homogenization introduced in § 1. The cone Q considered in § 1 is just H/I.

From these considerations we can infer the following theorem, due (in a slightly different form) to Sherali and Adams [22].

THEOREM 3.3. *If* $\mathcal{F} \subseteq 2^S$ *then* $\text{conv}\,\{\chi^A : A \in \mathcal{F}\}$ *is the projection of the following cone to singleton sets*:

$$x_\emptyset = 0, \qquad \mu_j^T x \geqq 0 \qquad (j \in \mathcal{F}), \qquad \mu_j^T x = 0 \qquad (j \notin \mathcal{F}).$$

The $(n \geqq 1) \times (n+1)$ matrices Y used in § 1 can be viewed in this framework in two different ways. First, they can be viewed as portions of the vector $x \in \mathbb{R}^{2^S}$ determined by the entries indexed by $\emptyset$, singletons, and pairs; the linear constraints on $M(K)$ used in § 1 are only the constraints we can derive in a natural way from the constraints involving just the first $n+1$ variables.

Second, the matrices Y also occur as principal minors of the corresponding (huge) matrix W^x. So the positive semidefiniteness constraint for $M_+(K)$ is just a relaxation of the condition that for $x \in H$, W^x is positive semidefinite. (It is interesting to observe that while by Corollary 3.2, the positive semidefiniteness of W^x is a polyhedral condition, this relaxation of it is not.)

Let us discuss the case of the stable set polytope. We have a graph $G = (V, E)$ and we take $S = V$, $L = 2^S$. Let F consist of the stable sets of G. Then cone $(F) \subseteq \mathbb{R}^L$ is defined by the constraints

$$x \in H, \qquad x_{ij} = 0 \quad \text{for every } ij \in E.$$

We can relax the first constraint by stipulating that the upper left $(n+1) \times (n+1)$ submatrix W_0^x of W^x is positive semidefinite. Then these submatrices form exactly the cone M_{TH} as introduced in § 2. As we have seen, the projection of this cone to $\mathbb{R}^I$, intersected with the hyperplane $x_0 = 1$, gives the body TH (G).

Note that the "supermodularity" constraints $x_{ij} - x_i - x_j + x_0 \geqq 0$ are linear constraints valid for H, and involve only the variables indexed by sets with cardinality at most 2, but they do not follow from the positive semidefiniteness of W_0^x. Using these inequalities we obtain from $x_{ij} = 0$ the constraint $x_i \geqq x_j \leqq x_0$ for every edge $ij \in E$.

Returning to our general setting, we are going to interpret the operators N, N_+, and $\hat{N}$ in this general setting, using the group algebra. In order to describe the projection of cone (F) on $\mathbb{R}^I$, we want to generate linear constraints valid for cone (F) such that only the coefficients corresponding to elements of I are nonzero. To this end, we use the semigroup algebra to combine constraints to yield new constraints for cone (F). (This may temporarily yield constraints having some further nonzero coefficients, which we can eliminate afterwards.)

We have already seen that $a \vee a \in \text{cone}\,(F)^*$ for every a. From (2) and (6) we can read off the following further rules:

(a) If $a, b \in \text{cone}\,(F)^*$, then $a \vee b \in \text{cone}\,(F)^*$.

(b) If $a \in \text{int}\,(\text{cone}\,(F)^*)$ and $a \vee b \in \text{cone}\,(F)^*$, then $b \in \text{cone}\,(F)^*$.

In rule (b), we can replace the condition that $a \in \text{int}\,(\text{cone}\,(F)^*)$ by the perhaps more manageable condition that $a = e_0 + c$ with $c \in \text{cone}\,(F)^*$. In fact, $e_0 \in \text{int}\,(\text{cone}\,(F)^*)$ and hence for every $c \in \text{cone}\,(F)^*$, $e_0 + c \in \text{int}\,(\text{cone}\,(F)^*)$. Conversely, if $a \in \text{int}\,(\text{cone}\,(F)^*)$, then for a sufficiently small $t > 0$, $a - te_0 \in \text{cone}\,(F)^*$. Set $c = (a - e_0)/t$, then $c + e_0 \in \text{cone}\,(F)^*$ and $(c + e_0) \vee b = (a \vee b)/t \in \text{cone}\,(F)^*$, and hence $b \in \text{cone}\,(F)^*$.

If $Z^T a > 0$, then rule (b) follows from rule (a). In fact, let $c(k) = 1/(Z^T a)(k)$, and $d = M^T c$. Then d is the inverse of a, that is, $d \vee a = e_0$, and $(Z^T d)(k) = c(k) > 0$ for all k, so $d \in \text{cone}\,(F)^*$. Hence

$$b = (a \vee b) \vee d \in \text{cone}\,(F)^*,$$

by rule (a).

For two cones $K_1, K_2 \subseteq \mathbb{R}^L$, we denote by $K_1 \vee K_2$ the cone spanned by all vectors $u_1 \vee u_2$, where $u_i \in K_i$. (The set of all vectors arising in this way is not convex in general.) This operation generalizes the construction of $N(K_1, K_2)$, $N_+(K_1, K_2)$, and $\hat{N}(K)$ in the following sense.

PROPOSITION 3.4. *Let $L = 2^S$, I, the set consisting of $\emptyset$, and the singleton subsets of S, and let $K_1, K_2 \subseteq H/I$ be two convex cones. Then*

(i) $N(K_1, K_2)^* = ((K_1^*)_I \vee (K_2^*)_I)_I$;

(ii) $N_+(K_1, K_2)^* = ((K_1^*) \vee (K_2^*)_I + \mathbb{R}^I \vee \mathbb{R}^I)_I$.

Proof of (i). First, we assume that $w \in ((K_1^*)_I \vee (K_2^*)_I)_I$. Then we can write $w = \sum_t a_t \vee b_t$, where $a_t \in (K_1^*)_I$ and $b_t \in (K_2^*)_I$. Let $x \in N(K_1, K_2)$; then we can write $x = Ye_0$ with $Y = (y_{ij} \in M(K_1, K_2))$. Define the vector $y \in \mathbb{R}^L$ by

$$y(k) = \begin{cases} x_k, & \text{if } k \in I, \\ y_{ij}, & \text{if } k = \{y, j\}, \\ 0, & \text{else.} \end{cases}$$

Then we have

$$w^T x = w^T y = \sum_t (a_t \vee b_t)^T y = \sum_t a_t^T Y b_t \geqq 0.$$

This proves that $w \in N(K_1, K_2)^*$.

Second, assume that $w \in N(K_1, K_2)^*$. Then we can write

$$we_0^T = \sum_t a_t b_t^T + \sum_{i=1}^n \lambda_i e_i f_i^T + A,$$

where $a_t \in K_1^*$, $b_t \in K_2^*$, $\lambda_i \in \mathbb{R}$, and A is a skew symmetric matrix. Now it is easy to

check that

$$w = \sum_t (a_t \vee b_t),$$

and so $w \in ((K_1^*)_I \vee (K_2^*)_I)_I$.

The proof of part (ii) is analogous. □

Next we show that the construction of $\hat{N}$ is, in fact, a special case of the application of rule (b).

LEMMA 3.5. *Let $L = 2^S$, I, the set consisting of $\emptyset$, and the singleton subsets of S, and let $K \subseteq H/I$, a convex cone. Then*

$$\hat{N}(K)^* = \{a \in \mathbb{R}^I : \exists b \in \operatorname{int}(K^*)_I \text{ such that } a \vee b \in (K^*)_I \vee (Q^*)_I\}.$$

The proof is analogous to that of Proposition 3.3, and is omitted.

We can use the formula in Proposition 3.4 to formulate a stronger version of the repetition of the operator N. Note that

$$N^2(K)^* = [[(K^*)_I \vee (Q^*)_I]_I \vee (Q^*)_I]_I \subseteq [(K^*)_I \vee (Q^*)_I \vee (Q^*)_I]_I,$$

and similarly, if we denote $(Q^*)_I \vee \cdots \vee (Q^*)_I$ (r factors) by Q_r, then

$$N^r(K)^* \subseteq [(K^*)_I \vee Q_r]_I.$$

Now it is easy to see that the cone Q_r is spanned by the vectors $\mu_{[i,j]}$ where $i \subseteq j$ and $|j| \leqq r$. For fixed r, this is a polynomial number of vectors. Let $\bar{N}^r(K)$ denote the polar cone of $[(K^*)_I \vee Q_r]_I$ in the linear space $\mathbb{R}^I$. Then $\bar{N}^r(K) \subseteq N^r(K)$.

For the case of boolean algebras (and in a quite different form), the sequence $\bar{N}^r(K)$ of relaxations of K^o was introduced by Sherali and Adams [22], who also showed that $\bar{N}^n(K) = K^o$.

It is easy to see that if K is polynomial time separable, then so is $\bar{N}^r(K)$ for every fixed r: to check whether $x \in \bar{N}^r(K)$, it suffices to check whether there exist vectors $a^{[i,j]} \in (K^*)_I$ for every i and j with $i \subseteq j$ and $|n| \leqq r$ such that $a = \sum_{i,j} a^{[i,j]} \vee \mu_{[i,j]} \in \mathbb{R}^I$ and $a^T x < 0$. This is easily done in polynomial time using the ellipsoid method.

Acknowledgments. The first author is grateful to the Department of Combinatorics and Optimization of the University of Waterloo for its hospitality while this paper was being written. Discussions with Mike Saks and Bill Pulleyblank on the topic of the paper were most stimulating. The authors also thank the referees for their insightful remarks.

REFERENCES

[1] E. BALAS AND W. R. PULLEYBLANK (1983), *The perfect matchable subgraph polytope of a bipartite graph*, Networks, 13, pp. 495–516.

[2] ——— (1989), *The perfectly matchable subgraph polytope of an arbitrary graph*, Combinatorica, 9, pp. 321–327.

[3] M. O. BALL, W. LIU, AND W. R. PULLEYBLANK (1989), *Two terminal Steiner tree polyhedra*, in Contributions to Operations Research and Economics, B. Tulkens and H. Tulkens, eds., MIT Press, Cambridge, MA, pp. 251–284.

[4] F. BARAHONA (1988), *Reducing matching to polynomial size linear programming*, Res. Report CORR 88-51, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada.

[5] F. BARAHONA AND A. R. MAHJOUB (1987), *Compositions of graphs and polyhedra* II: *Stable sets*, Res. Report 87464-OR, Institut für Operations Research, Universität Bonn, Bonn, FRG.

[6] K. CAMERON AND J. EDMONDS (1989), *Coflow polyhedra*, preprint, University of Waterloo, Waterloo, Ontario, Canada.
[7] S. CERIA (1989), personal communication.
[8] V. CHVÁTAL (1973), *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete Math., 4, pp. 305-337.
[9] ——— (1975), *On certain polytopes associated with graphs*, J. Combin. Theory Ser. B, 13, pp. 138-154.
[10] J. EDMONDS (1965), *Maximum matching and a polyhedron with* 0-1 *vertices*, J. Res. Nat. Bur. Standards, 69B, pp. 125-130.
[11] R. E. GOMORY (1963), *An algorithm for integer solutions to linear programs*, in Recent Advances in Mathematical Programming, R. Graves and P. Wolfe, eds., McGraw-Hill, New York, pp. 269-302.
[12] M. GRÖTSCHEL, L. LOVÁSZ, AND A. SCHRIJVER (1981), *The ellipsoid method and its consequences in combinatorial optimization*, Combinatorica, 1, pp. 169-197.
[13] ——— (1986), *Relaxations of vertex packing*, J. Combin. Theory Ser. B, 40, pp. 330-343.
[14] ——— (1988), *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, Berlin, New York.
[15] B. LINDSTRÖM (1969), *Determinants on semilattices*, Proc. Amer. Math. Soc., 20, pp. 207-208.
[16] W. LIU (1988), *Extended formulations and polyhedral projection*, Ph.D. thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada.
[17] L. LOVÁSZ (1979), *Combinatorial Problems and Exercises*, Akadémiai Kiadó, Budapest, Hungary; North-Holland, Amsterdam, the Netherlands.
[18] L. LOVÁSZ AND M. D. PLUMMER (1986), *Matching Theory*, Akadémiai Kiadó, Budapest, Hungary; Elsevier, Amsterdam, the Netherlands.
[19] N. MACULAN (1987), *The Steiner problem in graphs*, Ann. Discrete Math., 31, pp. 185-222.
[20] R. PEMANTLE, J. PROPP, AND D. ULLMAN (1989), *On tensor powers of integer programs*, preprint.
[21] G.-C. ROTA (1964), *On the foundations of combinatorial theory* I. *Theory of Möbius functions*, Z. Wahrsch. Verw. Gebiete, 2, pp. 340-368.
[22] H. D. SHERALI AND W. P. ADAMS (1988), *A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems*, preprint.
[23] R. P. STANLEY (1986), *Enumerative Combinatorics*, Vol. 1, Wadsworth, Monterey, CA.
[24] H. S. WILF (1968), *Hadamard determinants, Möbius functions, and the chromatic number of a graph*, Bull. Amer. Math. Soc., 74, pp. 960-964.
[25] R. J. WILSON (1970), *The Selberg sieve for a lattice*, in Combinatorial Theory and Its Applications, P. Erdös, A. Rényi, and V. T. Sós, eds., Coll. Math. Soc. János Bolyai, North-Holland, Amsterdam, 4, pp. 1141-1149.
[26] M. YANNAKAKIS (1988), *Expressing combinatorial optimization problems by linear programs*, in Proc. 29th IEEE Symposium on Foundations of Computer Science, White Plains, NY, pp. 223-228.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 191-205, May 1991

003

CONVERGENCE OF BEST ENTROPY ESTIMATES*

J. M. BORWEIN† AND A. S. LEWIS‡

Abstract. Given a finite number of moments of an unknown density $\bar{x}$ on a finite measure space, the best entropy estimate—that nonnegative density x with the given moments which minimizes the Boltzmann-Shannon entropy $I(x) := \int x \log x$—is considered. A direct proof is given that I has the Kadec property in L_1—if y_n converges weakly to $\bar{y}$ and $I(y_n)$ converges to $I(\bar{y})$, then y_n converges to $\bar{y}$ in norm. As a corollary, it is obtained that, as the number of given moments increases, the best entropy estimates converge in L_1 norm to the best entropy estimate of the limiting problem, which is simply $\bar{x}$ in the determined case. Furthermore, for classical moment problems on intervals with $\bar{x}$ strictly positive and sufficiently smooth, error bounds and uniform convergence are actually obtained.

Key words. moment problem, entropy, Kadec, partially finite program, normal convex integrand, duality

AMS(MOS) subject classifications. primary 41A46, 05C38; secondary 08A45, 28A20

1. Introduction.

We shall suppose that (S, μ) is a finite measure space, and define the closed proper convex function $\phi : \mathbb{R} \to (-\infty, +\infty]$ by

$$\phi(u) := \begin{cases} u \log u, & \text{if } u > 0, \\ 0, & \text{if } u = 0, \\ +\infty, & \text{if } u < 0. \end{cases}$$

This function is a normal convex integrand [18], allowing us to define (minus) the Boltzmann-Shannon entropy $I_\phi(x) : L_1(S, \mu) \to (-\infty, +\infty]$ by

$$I_\phi(x) := \int_S \phi(x(s))\, d\mu(s). \tag{1}$$

Suppose $0 \leqq \bar{x} \in L_1(S, \mu)$ is an unknown density that we wish to estimate on the basis of a finite number of observed moments,

$$b_i = \int_S \bar{x}(s) a_i(s)\, d\mu(s), \qquad i = 1, \cdots, n,$$

where the a_i's are given functions in $L_\infty(S, \mu)$. This is a problem which commonly arises in diverse areas of physics, engineering, and statistics (see, for example, [14] and [11]). One popular technique is to choose the maximum entropy estimate—the solution of the optimization problem

$$(\mathrm{P}_n) \qquad \begin{cases} \text{minimize} & I_\phi(x) \\ \text{subject to} & \displaystyle\int_S x(s) a_i(s)\, d\mu(s) = b_i, \qquad i = 1, \cdots, n, \\ & 0 \leqq x \in L_1(S, \mu). \end{cases}$$

Attractive dual methods are available for solving the problems (P_n) computationally (see, for example, [7]).

* Received by the editors August 9, 1990; accepted for publication (in revised form) October 10, 1990. This research was partially supported by the Natural Sciences and Engineering Research Council of Canada.

† Department of Mathematics, Statistics, and Computing Science, Dalhousie University, Halifax, Nova Scotia, Canada B3H 3J5.

‡ Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

One measure of the effectiveness of this approach is the behavior of the optimal solution x_n of (P_n) as $n \to \infty$ (see, for example, [5] and [20]). At least when $\bar{x}$ is determined uniquely by the moment sequence $(\int \bar{x} a_i)_1^\infty$ we would hope that x_n converged to $\bar{x}$ in some sense. In [17] it was shown essentially that in this case $x_n\, d\mu$ converged to $\bar{x}\, d\mu$ weak-star as measures, while in [3] this was strengthened to the result that x_n converges weakly to $\bar{x}$ in L_1 (see also [10] and [15]). In this paper we will show that, in fact, x_n converges in L_1 norm to $\bar{x}$, and that with some further assumptions, the convergence is actually uniform.

The results break naturally into two parts. In the first we demonstrate the simple but remarkable geometric fact that, in common with the L_p norms $(1 < p < \infty)$, the Boltzmann–Shannon entropy I_ϕ has the Kadec property: weak convergence of x_n to $\bar{x}$ and convergence of $I_\phi(x_n)$ to $I_\phi(\bar{x})$ implies norm convergence of x_n to $\bar{x}$. Our proof will be self-contained and straightforward. In the section which follows, we deduce the required convergence result and discuss some further implications.

In the second set of results, we begin by deriving a bound for the L_1 error in estimating $\bar{x}$ by x_n using duality techniques. Finally, when $\bar{x}$ is strictly positive and sufficiently smooth, we are able to combine this bound with ideas from the first collection of results and some standard approximation theory to show that, for classical algebraic and trigonometric moment problems on intervals, the best entropy estimates x_n converge **uniformly** to the underlying unknown density $\bar{x}$.

2. Strongly convex functions. In this section we derive the geometric property of the entropy which we will apply to the question of convergence.

LEMMA 2.1. *The Boltzmann–Shannon entropy I_ϕ defined in* (1) *is a proper, lower semicontinuous convex function, strictly convex on its domain,*

$$\operatorname{dom} I_\phi := \{x \in L_1(S, \mu) \mid I_\phi(x) < +\infty\},$$

and with weakly compact level sets, $\{x \in L_1(S, \mu) \mid I_\phi(x) \leqq \alpha\}$, for all $\alpha \in \mathbb{R}$.

Proof. (See [19].) The fact that the level sets are weakly compact follows either from the fact that the conjugate function $\phi^*(v) = e^{v-1}$ is everywhere finite, or directly from the Dunford–Pettis criterion [8]. □

The following inequality relating the so-called *I-divergence* of two probability densities with their L_1 distance appeared independently in [6], [12], and [13]. For completeness, we include a proof, following [12].

LEMMA 2.2 (a) *For $0 < v \in \mathbb{R}$, $0 \leqq u \in \mathbb{R}$,*

$$(u - v)^2 \leqq ((2u/3) + (4v/3))(u \log(u/v) - u + v).$$

(b) *For $0 \leqq x, y \in L_1(S, d\mu)$ with $\int_S x(s)\, d\mu = \int_S y(s)\, d\mu = 1$,*

$$\int_S x(s) \log(x(s)/y(s))\, d\mu \geqq \frac{1}{2}\left(\int_S |x(s) - y(s)|\, d\mu\right)^2.$$

Proof. (a) It is easy to check by differentiating that the function

$$\left(\frac{4}{3} + \frac{2t}{3}\right)(t \log t - t + 1) - (t - 1)^2$$

is convex on $[0, +\infty)$, and attains a minimum value of 0 when $t = 1$. Putting $t := u/v$ gives (a).

(b) If $y(s) = 0$ and $x(s) > 0$ simultaneously on a nonnull set, then the left-hand side is $+\infty$ and there is nothing to prove. Therefore, assuming this does not occur, we

can restrict attention to the case where $x(s)>0$ and $y(s)>0$ almost everywhere (if necessary, removing the set where $x(s)=y(s)=0$). Now set $u:=x(s)$ and $v:=y(s)$ in (a), take square-roots, and apply the Cauchy-Schwarz inequality to obtain (b), noting that $u\log(u/v)-u+v\geqq 0$, by (a). □

Notes. (1) We use the inequality (b) only in the case where $\log y\in L_\infty$, in which case there is no difficulty in defining the left-hand side in (b). If, however, we wish to be more precise, we define the left-hand side as $+\infty$ unless $x\geqq 0$; otherwise it is defined as

$$I_\phi(x)+\int_S \psi(y(s),s)\,d\mu(s),$$

where $\psi: R\times S\to(-\infty,+\infty]$ is the normal convex integrand

$$\psi(v,s):=\begin{cases}0, & \text{if } x(s)=0,\\ -x(s)\log v, & \text{if } x(s)>0, \quad v>0,\\ +\infty, & \text{if } x(s)>0, \quad v\leqq 0\end{cases}$$

(see [18]).

(2) It is easy to see from the proof that inequality (b) is strict unless $x=y$ almost everywhere.

(3) As observed in [12], the constant $\frac{1}{2}$ on the right in (b) is the best possible.

(4) We cannot hope for a similar inequality with $\|x-y\|_p^2$ replacing $\|x-y\|_1^2$ for some $p>1$ on the right. To see this, take $S=[0,1]$ with Lebesgue measure; define, for $k>1$,

$$x(s):=\begin{cases}k, & \text{for } s\in[0,1/k],\\ 0, & \text{for } s\in(1/k,1];\end{cases}$$

and let $y(s):=1$ almost everywhere. Then $\int x\log x/y=\log k$, while it is easy to see that $\|x-y\|_p^2\sim k^{2(1-1/p)}$.

Despite observation (4) above we can prove a somewhat similar result for the L_p norms.

PROPOSITION 2.3. *For* $0\leqq x,\ y\in L_\infty(S,d\mu)$ *with* $\int_S x(s)\,d\mu=\int_S y(s)\,d\mu$, *and* $2\leqq p<+\infty$,

$$\int_S x(s)\log(x(s)/y(s))\,d\mu\geqq(1/p(p-1))(\max\{\|x\|_\infty,\|y\|_\infty\})^{1-p}\|x-y\|_p^p.$$

Proof. Suppose $v\in(0,1]$. It is easy to check by differentiation that

$$\varphi(u):=u\log(u/v)-u+v-(1/p(p-1))|u-v|^p$$

is convex for $u\in[0,1]$ and attains a minimum value of 0 at $u=v$. Thus for $u\in[0,1]$, $v\in(0,1]$, we have

$$u\log(u/v)-u+v\geqq(1/p(p-1))|u-v|^p.$$

As before, we can restrict attention to the case where $y(s)>0$, almost everywhere. Now if we set $M:=\max\{\|x\|_\infty,\|y\|_\infty\}$, put $u:=(1/M)x(s)$ and $v:=(1/M)y(s)$ in the above, and integrate, we obtain the result. □

We are now ready to prove the geometric property of the Boltzmann-Shannon entropy I_ϕ that we will apply to the moment problem. In keeping with the terminology for normed spaces we make the following definition.

DEFINITION 2.4. Suppose X is a normed space with $f: X \to (-\infty, +\infty]$.

(a) (See [3].) The function f is **Kadec** if, whenever $y_n \to \bar{y}$ weakly in X and $f(y_n) \to f(\bar{y}) < +\infty$, it follows that $y_n \to \bar{y}$ in norm.

(b) The function f is **strongly convex** if it is Kadec and is a lower semicontinuous convex function, strictly convex on its domain $\{x | f(x) < +\infty\}$, with weakly compact level sets $\{x | f(x) \leqq \alpha\}$ for $\alpha \in \mathbb{R}$.

For example, with $X = L_p(S, \mu)$ for $1 < p < +\infty$, the norm $f(x) := \|x\|_p$ is strongly convex (see, for example, [8]). The main result of this section will be that the entropy I_ϕ is strongly convex on $L_1(S, \mu)$. By Lemma 2.1, it remains to show that I_ϕ is Kadec.

LEMMA 2.5. *Suppose* $0 \leqq y_n$, $\bar{y} \in L_1(S, \mu)$, *for* $n = 1, 2, \cdots$, *with, for each* n, $\int y_n = \int \bar{y} > 0$. *Suppose further that* $y_n \to \bar{y}$ *weakly in* L_1 *and that* $I_\phi(y_n) \to I_\phi(\bar{y}) < +\infty$. *Then* $\|y_n - \bar{y}\|_1 \to 0$.

Proof. By scaling the measure μ by a scalar factor, we lose no generality by supposing that $\int y_n = \int \bar{y} = 1$, for all n. For $m = 1, 2, \cdots$, we write $(x \wedge m)(s) := \min\{x(s), m\}$, and define

$$y^m := (1 - (1/m))\left[\left(\int \bar{y} \wedge m\right)^{-1} (\bar{y} \wedge m)\right] + (1/m)[\mu(S)^{-1}]. \tag{2}$$

Thus $\log y^m \in L_\infty$, and $\int y^m = 1$.

Now we have

$$\begin{aligned}
\frac{1}{2}\|y_n - y^m\|_1^2 &\leqq \int y_n \log(y_n / y^m), \quad \text{by Lemma 2.2,} \\
&= I_\phi(y_n) - \int y_n \log y^m \\
&\to I_\phi(\bar{y}) - \int \bar{y} \log y^m, \quad \text{as } n \to \infty, \text{ by assumption,} \\
&\leqq -\int \bar{y}\left\{(1 - (1/m)) \log\left[\left(\int \bar{y} \wedge m\right)^{-1} (\bar{y} \wedge m)\right] \cdots \right. \\
&\qquad \left. + (1/m) \log[\mu(S)^{-1}]\right\} + I_\phi(\bar{y}) \quad \text{(convexity and (2))} \\
&= (1 - (1/m))\left[\left\{\log\left(\int \bar{y} \wedge m\right)\right\}\left\{\int \bar{y}\right\} - \int \bar{y} \log(\bar{y} \wedge m)\right] \\
&\qquad + (1/m) \log[\mu(S)]\left\{\int \bar{y}\right\} + I_\phi(\bar{y}).
\end{aligned}$$

Now since $u \log u \geqq -1/e$ for any $u \geqq 0$, we have

$$-1/e \leqq \bar{y} \log(\bar{y} \wedge m) \uparrow \bar{y} \log \bar{y} \in L_1,$$

as $m \to \infty$, so we can apply the monotone convergence theorem to deduce that as $m \to \infty$, the right-hand side tends to 0. Thus we obtain

$$\lim_{m \to \infty} \overline{\lim_{n \to \infty}} \|y_n - y^m\|_1 = 0. \tag{3}$$

We also have

$$\begin{aligned}\|y^m-\bar{y}\|_1 &= \left\|(1-(1/m))\left[\left(\int \bar{y}\wedge m\right)^{-1}(\bar{y}\wedge m)\right]+(1/m)[\mu(S)^{-1}]-\bar{y}\right\|_1\\ &\leqq (1/m)+\left\|(1-(1/m))\left(\int \bar{y}\wedge m\right)^{-1}[(\bar{y}\wedge m)-\bar{y}]\right\|_1\\ &\quad+\left\|\left[(1-(1/m))\left(\int \bar{y}\wedge m\right)^{-1}-1\right]\bar{y}\right\|_1\\ &= (1/m)+(1-(1/m))\left(\int \bar{y}\wedge m\right)^{-1}\int[\bar{y}-(\bar{y}\wedge m)]\\ &\quad+\left[(1-(1/m))\left(\int \bar{y}\wedge m\right)^{-1}-1\right]\int \bar{y}\\ &\to 0, \quad \text{as } m\to\infty \text{ by monotone convergence.}\end{aligned}$$

Finally, by combining (3) and the above, we obtain

$$\overline{\lim_{n\to\infty}}\,\|y_n-\bar{y}\|_1 \leqq \overline{\lim_{n\to\infty}}\,\|y_n-y^m\|_1+\|y^m-\bar{y}\|_1\to 0, \quad \text{as } m\to 0. \qquad \square$$

LEMMA 2.6. *Suppose X is a normed space, and $\alpha_n\to\bar{\alpha}$ in $\mathbb{R}$.*
(a) *If $w_n\to\bar{w}$ weakly in X, then $\alpha_n w_n\to\bar{\alpha}\bar{w}$ weakly.*
(b) *If $\|w_n-\bar{w}\|\to 0$, then $\|\alpha_n w_n-\bar{\alpha}\bar{w}\|\to 0$.*
Proof. The proof is elementary. $\square$

THEOREM 2.7. *The Boltzmann–Shannon entropy I_ϕ is strongly convex.*

Proof. By Lemma 2.1, we just have to show that I_ϕ is Kadec. To this end, suppose $z_n, \bar{z}\in L_1$, for $n=1,2,\cdots$; $z_n\to\bar{z}$ weakly in L_1; and $I_\phi(z_n)\to I_\phi(\bar{z})<+\infty$. It follows that $I_\phi(z_n)<+\infty$, for all n sufficiently large, so $\bar{z}, z_n\geqq 0$.

Consider first the case where $\bar{z}\neq 0$, so $\int \bar{z}>0$. By weak convergence, $\int z_n\to\int \bar{z}$, so for all n sufficiently large, $\int z_n>0$, and we can define functions $\bar{y}:=(\int \bar{z})^{-1}\bar{z}$ and $y_n:=(\int z_n)^{-1}z_n$. Thus $0\leqq y_n$, $\bar{y}\in L_1$, $\int y_n=\int \bar{y}$ for each n, and by Lemma 2.6, $y_n\to\bar{y}$ weakly. Furthermore,

$$\begin{aligned}I_\phi(y_n) &= \int\left\{\left(\int z_n\right)^{-1}z_n\log\left[\left(\int z_n\right)^{-1}z_n\right]\right\}\\ &= \left(\int z_n\right)^{-1}\left\{I_\phi(z_n)-\left(\log\int z_n\right)\int z_n\right\}\\ &\to \left(\int \bar{z}\right)^{-1}\left\{I_\phi(\bar{z})-\left(\log\int \bar{z}\right)\int \bar{z}\right\}\\ &= I_\phi(\bar{y}).\end{aligned}$$

Thus Lemma 2.5 applies to show that $\|y_n-\bar{y}\|_1\to 0$, so by Lemma 2.6, $\|z_n-\bar{z}\|_1\to 0$.

Finally, suppose $\bar{z}=0$. Since $z_n\geqq 0$ for large n, we have, as $n\to\infty$, $\|z_n-\bar{z}\|_1=\int z_n\to\int \bar{z}=0$ by weak convergence. $\square$

If we know a priori that the sequence (z_n) in the above proof of the Kadec property is uniformly bounded, we obtain a much stronger conclusion.

THEOREM 2.8. *Suppose $0\leqq z_n\leqq M$ almost everywhere, $z_n\to\bar{z}$ weakly in L_1, and $I_\phi(z_n)\to I_\phi(\bar{z})$. Then for any $p<+\infty$, $\|z_n-\bar{z}\|_p\to 0$ as $n\to\infty$.*

Proof. We first note that since the positive cone in L_1 is closed, and hence weakly closed, it follows from the assumptions that $0\leqq\bar{z}\leqq M$, almost everywhere.

The proof is now exactly analogous to Lemma 2.5 and Theorem 2.7, with minor changes. We can simplify the definition of y^m in (2) to

$$y^m := (1-(1/m))\bar{y} + (1/m)[\mu(S)]^{-1},$$

and we use Proposition 2.3 in place of Lemma 2.2. The only real change is the case $\bar{z}=0$ in Theorem 2.7. For large n we know $z_n \geqq 0$, and we may as well assume that $z_n \neq 0$. Now we can assert, by Proposition 2.3, for $p \geqq 2$, large n, and some $M_1 \geqq M_2$

$$(1/p(p-1))M_1^{1-p} \left\| z_n - \int z_n \right\|_p^p \leqq \int z_n \log\left(z_n \Big/ \int z_n\right)$$

$$= I_\phi(z_n) - \left(\int z_n\right) \log\left(\int z_n\right) \to 0, \quad \text{as } n \to \infty,$$

from which it follows that $\|z_n\|_p \to 0$. □

3. L_1 convergence. In this section we will apply the strong convexity of the entropy I_ϕ to deduce, in particular, that if the unknown density $\bar{x}$ is uniquely determined by its moments, $(\int \bar{x}a_i)_{i=1}^\infty$, then the optimal solution x_n of (P_n) converges in L_1 norm to $\bar{x}$. The approach will be through the following elementary result, which may be found in [3].

THEOREM 3.1. *Let X be a topological space, with a nested sequence of closed subsets, $X \supset F_1 \supset F_2 \supset \cdots$, and suppose $f: X \to (-\infty, +\infty]$ has compact level sets. Consider the optimization problems*

$$(Q_n) \qquad \inf\{f(x) \mid x \in F_n\}$$

$$(Q_\infty) \qquad \inf\left\{f(x) \mid x \in \bigcap_{n=1}^\infty F_n\right\}.$$

The values of (Q_n) and (Q_∞) are attained, if finite, and the value of (Q_n) increases in n to the value of (Q_∞) (finite or infinite). Suppose furthermore that x_n is optimal for (Q_n), and x_∞ is the unique optimal solution for (Q_∞), with finite value. Then $x_n \to x_\infty$.

COROLLARY 3.2. *Let X be a normed space with a nested sequence of closed, convex subsets, $X \supset F_1 \supset F_2 \supset \cdots$, and suppose $f: X \to (-\infty, +\infty]$ is strongly convex. Suppose that (Q_∞) has finite value. Then (Q_n) and (Q_∞) have unique optimal solutions (with finite value), x_n and x_∞, respectively, and $x_n \to x_\infty$ in norm.*

Proof. Existence follows from Theorem 3.1, and uniqueness is a consequence of strict convexity. Theorem 3.1 shows that $x_n \to x_\infty$ weakly, and also $f(x_n) \to f(x_\infty)$, whence $x_n \to x_\infty$ in norm, by the Kadec property. □

We recall the problems (P_n) of § 1:

$$(P_n) \qquad \begin{cases} \text{minimize} & I_\phi(x) \\ \text{subject to} & \displaystyle\int_S x(s)a_i(s)\,d\mu(s) = b_i, \qquad i = 1, \cdots, n, \\ & 0 \leqq x \in L_1(S, \mu). \end{cases}$$

The limiting problem is

$$(P_\infty) \qquad \begin{cases} \text{minimize} & I_\phi(x) \\ \text{subject to} & \displaystyle\int_S x(s)a_i(s)\,d\mu(s) = b_i, \qquad i = 1, 2, \cdots, \\ & 0 \leqq x \in L_1(S, \mu). \end{cases}$$

Applying Corollary 3.2 gives the following result.

COROLLARY 3.3. *The value of* (P_n) *increases in n to the value of* (P_∞) (*finite or infinite*). *If* (P_∞) *has finite value, then* (P_n) *and* (P_∞) *have unique optimal solutions* (*with finite value*), x_n *and* x_∞, *respectively, and* $\|x_n - x_\infty\|_1 \to 0$.

Proof. The proof is by Corollary 3.2. □

Notes. (1) Assuming, as in § 1, that the b_i's are the moments of an unknown density $0 \leqq \bar{x} \in L_1$, then if $I_\phi(\bar{x}) < +\infty$ it follows that (P_∞) has finite value.

(2) If, furthermore, S is a compact metric space with Borel measure μ, and the linear span of $\{a_i \mid i = 1, 2, \cdots\}$ is dense in the continuous functions $C(S)$ (as in the classical trigonometric and algebraic moment problems), then it is easily checked that $\bar{x}$ is uniquely determined by its moments $(\int \bar{x} a_i)$, and so $\bar{x} = x_\infty$. In this case $\|x_n - \bar{x}\|_1 \to 0$.

(3) Convergence in L_1 norm is the best possible: in general, we cannot expect convergence in L_p norm for any $p > 1$. To see this, suppose $0 \leqq \bar{x} \in L_1$ with $I_\phi(\bar{x}) < +\infty$, but that $\bar{x} \notin L_p$ for any $p > 1$. Such functions are not difficult to construct (see, for example, [21]). It is well known that, under mild assumptions (see Theorem 4.2), the unique optimal solution of (P_n) is of the form

$$x_n = e^{\sum_{i=1}^n \lambda_i a_i - 1}$$

for some $\lambda \in \mathbb{R}^n$, and so $x_n \in L_\infty$ for each n. If $\|x_n - \bar{x}\|_p \to 0$, it would follow that $\bar{x} \in L_p$, which is a contradiction for $p > 1$.

(4) On the other hand, if the x_n's are uniformly bounded, then we can apply Theorem 2.8 in place of the Kadec property to deduce that $\|x_n - \bar{x}\|_p \to 0$ as $n \to \infty$, for every $p < +\infty$. Unfortunately it is unclear how we might know uniform boundedness of $(x_n)_1^\infty$ a priori. We return to this question of stronger convergence in the next section.

In some estimation problems it is natural to suppose that the unknown density $\bar{x}$ is bounded above by some known constant $0 < K \in \mathbb{R}$ (see, for example, [7]). In this case it may be appropriate to modify the Boltzmann-Shannon entropy $I_\phi(x)$ to $I_\phi(x) + I_\phi(K - x)$, thereby incorporating this information. We then arrive at the following modified problems:

$$(P_n^K) \qquad \begin{cases} \text{minimize} & I_\phi(x) + I_\phi(K - x) \\ \text{subject to} & \displaystyle\int_S x(s) a_i(s)\, d\mu(s) = b_i, \qquad i = 1, \cdots, n, \\ & x \in L_1(S, \mu), \end{cases}$$

and the limiting problem

$$(P_\infty^K) \qquad \begin{cases} \text{minimize} & I_\phi(x) + I_\phi(K - x) \\ \text{subject to} & \displaystyle\int_S x(s) a_i(s)\, d\mu(s) = b_i, \qquad i = 1, 2, \cdots, \\ & x \in L_1(S, \mu). \end{cases}$$

The following proposition concerning strong convexity is useful in this context.

PROPOSITION 3.4. *Let* X *be a normed space with* $f, g : X \to (-\infty, +\infty]$. *Suppose* f *is strongly convex and* g *is convex, lower semicontinuous, and bounded below. Then* $f + g$ *is strongly convex.*

Proof. Clearly $f+g$ is lower semicontinuous and convex, since f and g are, and is strictly convex on its domain, since f is. Suppose $g \geqq M$. Then the level set

$$\{x | (f+g)(x) \leqq \alpha\} \subset \{x | f(x) \leqq \alpha - M\},$$

and is closed, so therefore it is weakly compact. Finally, the fact that $f+g$ is Kadec follows from Theorem 6.5 in [3]. □

From Corollary 3.2 we immediately deduce that if (P_∞^K) has finite value, then the unique optimal solution x_n' of (P_n^K) converges in L_1 norm to the unique optimal solution x_∞' of (P_∞^K) (and corresponding comments to Notes 1 and 2 following Corollary 3.3 hold). However, we can prove a stronger result.

THEOREM 3.5. *The value of* (P_n^K) *increases in n to the value of* (P_∞^K) (*finite or infinite*). *If* (P_∞^K) *has finite value, then* (P_∞^K) *and* (P_n^K) *have unique optimal solutions* (*with finite value*) x_∞' *and* x_n', *respectively, and* $\|x_n' - x_\infty'\|_p \to 0$, *as* $n \to \infty$, *for every* $p < +\infty$.

Proof. Since $\phi(u) \geqq -1/e$ for all u, $I_\phi(K-x)$ is bounded below (and certainly is convex and lower semicontinuous). Therefore, by Proposition 3.4, $I_\phi(x) + I_\phi(K-x)$ is strongly convex, so we can apply Theorem 3.1 and Corollary 3.2 to deduce the first assertions and the fact that $\|x_n' - x_\infty'\|_1 \to 0$. Thus by lower semicontinuity, $\underline{\lim}_{n\to\infty} I_\phi(x_n') \geqq I_\phi(x_\infty')$. However, we also know that

$$\begin{aligned}
\overline{\lim_{n\to\infty}} I_\phi(x_n') &= \overline{\lim}\,(I_\phi(x_\infty') + I_\phi(K - x_\infty') - I_\phi(K - x_n')) \\
&= I_\phi(x_\infty') + I_\phi(K - x_\infty') - \underline{\lim_{n\to\infty}} I_\phi(K - x_n') \\
&\leqq I_\phi(x_\infty') + I_\phi(K - x_\infty') - I_\phi(K - x_\infty') \\
&= I_\phi(x_\infty'),
\end{aligned}$$

again by lower semicontinuity.

Thus $I_\phi(x_n') \to I_\phi(x_\infty')$, and Theorem 2.8 now gives the result. □

4. Error bounds and uniform convergence. In the last section we saw that the unique optimal solution x_n of the problem (P_n) converged in L_1 norm to the unique optimal solution of the limiting problem (P_∞) (which in the determined case is exactly the unknown density $\bar{x}$). In this section we will demonstrate how, in more special circumstances, we can provide bounds on the L_1 error between x_n and $\bar{x}$. In classical cases this in turn allows us to prove that when $\bar{x}$ is strictly positive and sufficiently smooth, x_n actually must converge uniformly to $\bar{x}$. This of course is the most desirable result in practice.

In order to accomplish this, we use a combination of ideas from the previous sections and results from classical approximation theory to investigate the relationship between (P_n) and its dual problem. We therefore begin by summarizing what is known in general about this duality (see, for example, [2]). Recall that the primal problem is

$$(P_n) \qquad \begin{cases} \text{minimize} & I_\phi(x) \\ \text{subject to} & \displaystyle\int_S (x - \bar{x}) a_i \, d\mu = 0, \qquad i = 1, \cdots, n, \\ & 0 \leqq x \in L_1(S, \mu). \end{cases}$$

The corresponding dual problem is then

$$(D_n) \qquad \begin{cases} \text{maximize} & \displaystyle\int_S \left(\bar{x} \left[\sum_{i=1}^n \lambda_i a_i \right] - e^{[\sum_{i=1}^n \lambda_i a_i] - 1} \right) d\mu \\ \text{subject to} & \lambda \in \mathbb{R}^n. \end{cases}$$

The following weak duality result is elementary.

PROPOSITION 4.1. *The value of* (P_n) *is greater than or equal to the value of* (D_n).

In order to claim equality between the values of the primal and dual problems, we need a constraint qualification:

(CQ) There exists an $\hat{x} \in L_1$, feasible for (P_n) with finite value, and with $\hat{x}(s) > 0$ a.e.

In practice, it is frequently the case that the constraint functions $\{a_1, \cdots, a_n\}$ are *pseudo-Haar* (in other words linearly independent on nonnull sets). For example, this is the case when the a_i's are linearly independent and analytic on a compact interval with Lebesgue measure (which covers the classical moment problems). In this case, providing that $\bar{x}$ is nonzero with finite value, (CQ) holds.

When the constraint qualification holds, we get a strong duality result.

THEOREM 4.2. *Suppose* (CQ) *holds. Then both* (P_n) *and* (D_n) *attain their values, which are equal. If* λ^n *is optimal for* (D_n), *then the unique optimal solution of* (P_n) *is*

$$x_n := e^{\sum_{i=1}^n \lambda_i^n a_i - 1}.$$

All these results may be found in [2].

We define the constant E_n associated with the problem (P_n) to measure how well it is possible to approximate $1 + \log \bar{x}$ uniformly with a linear combination of the a_i's, $i = 1, \cdots, n$.

DEFINITION 4.3. For each n, E_n is defined to be $+\infty$ unless $\log \bar{x} \in L_\infty$, in which case $E_n := \min \{\| \sum_{i=1}^n \lambda_i a_i - 1 - \log \bar{x} \|_\infty | \lambda \in \mathbb{R}^n\}$.

Using this constant we can now give a lower bound on the value of (D_n) (and therefore of (P_n)). We need the following lemma.

LEMMA 4.4. *Suppose* $\beta > 0$.

(a) *If* $|u| \leqq \beta$, *then* $1 + u \leqq e^u \leqq 1 + u + e^\beta \beta^2/2$.

(b) *If* $|v - w| \leqq \beta$, *then* $|e^v - e^w| \leqq \beta(1 + e^\beta \beta/2) e^w$.

Proof. (a) This part follows by Taylor's theorem, and convexity.

(b) $-\beta(1 + e^\beta \beta/2) < -\beta \leqq v - w \leqq e^{v-w} - 1 \leqq v - w + e^\beta \beta^2/2 \leqq \beta(1 + e^\beta \beta/2)$, by applying (a) twice. Thus $|e^{v-w} - 1| \leqq \beta(1 + e^\beta \beta/2)$, and the result now follows. □

THEOREM 4.5. *For every n we have*

$$I_\phi(\bar{x}) \geqq V(P_n) \geqq V(D_n) \geqq I_\phi(\bar{x}) - \frac{1}{2} e^{En} E_n^2 \int \bar{x},$$

where E_n *is given by Definition* 4.3 *and* $V(\cdot)$ *denotes value.*

Proof. The first inequality follows from the fact that $\bar{x}$ is always feasible for (P_n), while the second is Proposition 4.1. We need only check the last for $\log \bar{x} \in L_\infty$. Since span $\{a_1, \cdots, a_n\}$ is finite-dimensional, there exists $\bar{\lambda} \in \mathbb{R}^n$ attaining the minimum in Definition 4.3, so

$$\left| \sum_{i=1}^n \bar{\lambda}_i a_i - 1 - \log \bar{x} \right| \leqq E_n \quad \text{a.e.}$$

Applying Lemma 4.4(a) now gives

$$e^{\sum_{i=1}^n \bar{\lambda}_i a_i - 1 - \log \bar{x}} \leqq 1 + \sum_{i=1}^n \bar{\lambda}_i a_i - 1 - \log \bar{x} + \frac{1}{2} e^{E_n} E_n^2,$$

so multiplying by $\bar{x}$ (which is nonnegative) gives

$$e^{\sum_{i=1}^n \bar{\lambda}_i a_i - 1} - \bar{x} \sum_{i=1}^n \bar{\lambda}_i a_i \leqq -\bar{x} \log \bar{x} + \frac{1}{2} e^{E_n} E_n^2 \bar{x}.$$

Integrating now gives the result. □

The following assumption holds in most of the cases in which we are interested.

ASSUMPTION. $a_1 \equiv 1$.

PROPOSITION 4.6. *Suppose* $a_1 \equiv 1$. *Whenever* (P_n) *has finite value, denote its (unique) optimal solution by* x_n.

(a) *If* x^0 *is feasible for* (P_n) *with finite value, then* $\int x_n \log x_n \leqq \int x^0 \log x_n$. *In particular, if* (P_m) *has finite value, with* $m \geqq n$, *then we have that* $\int x_n \log x_n \leqq \int x_m \log x_n$, *and if* $I_\phi(\bar{x}) < +\infty$, *then it follows that* $\int x_n \log x_n \leqq \int \bar{x} \log x_n$.

(b) *Suppose* (CQ) *holds for* (P_n). *If* x^0 *is feasible for* (P_n), *then we have* $\int x_n \log x_n = \int x^0 \log x_n$. *In particular,* $\int x_n \log x_n = \int \bar{x} \log x_n$, *and if* (P_m) *has finite value, with* $m \geqq n$, *then* $\int x_n \log x_n = \int x_m \log x_n$.

Proof. Since ϕ is convex, it is easy to check that if $u > 0$,

$$(1/\nu)\{\phi(u+\nu w) - \phi(u)\} \downarrow (\log u + 1)w \tag{4}$$

as $\nu \downarrow 0$. Then since x_n is optimal for (P_n), we have

$$0 \leqq \lim_{\nu \downarrow 0} (1/\nu)\{I_\phi(x_n + \nu(x^0 - x_n)) - I_\phi(x_n)\}$$

$$= \int (\log x_n + 1)(x^0 - x_n) = \int (x^0 - x_n) \log x_n,$$

by the monotone convergence theorem (observing the fact that when $\nu = 1$, the integrand in the first inequality is integrable), providing $x_n > 0$ almost everywhere, which gives (a) in this case.

In view of Theorem 4.2, this is all we will use. However, in point of fact a more precise argument shows that $x_n(s) = 0$ implies $x^0(s) = 0$ almost everywhere (see [4]), allowing us to restrict the range of integration to $\{s \mid x_n(s) > 0\}$.

To see (b) we simply have to rewrite $\log x_n$ using the known form of the solution from Theorem 4.2. □

By combining the above result with the weak duality bound in Theorem 4.5, and using the inequality in Lemma 2.2, we obtain a bound on the L_1 error of x_n from $\bar{x}$ in terms of the approximation error E_n of Definition 4.3. Ignoring the case $\bar{x} = 0$, we lose no generality (scaling if necessary) in assuming $\int \bar{x} = 1$.

THEOREM 4.7. *Suppose* $a_1 \equiv 1$, $\int \bar{x} = 1$, *and* $I_\phi(\bar{x}) < +\infty$. *Then the optimal solution* x_n *satisfies*

$$\|x_n - \bar{x}\|_1 \leqq E_n e^{E_n/2}, \tag{5}$$

where E_n *is given by Definition* 4.3.

Furthermore, if $\bar{x} \in L_\infty$ *and the sequence* $(x_n)_1^\infty$ *is uniformly bounded in* L_∞, *then given any* $2 \leqq p < +\infty$,

$$\|x_n - \bar{x}\|_p^p \leqq K_p E_n^2 e^{E_n}, \tag{6}$$

where the constant K_p *is independent of* n.

Proof. By Proposition 4.6,

$$\int \bar{x} \log x_n \geqq \int x_n \log x_n = V(P_n) \geqq \int \bar{x} \log \bar{x} - \frac{1}{2} e^{E_n} E_n^2$$

by Theorem 4.5. Applying Lemma 2.2(b) gives

$$\frac{1}{2}\|x_n - \bar{x}\|_1^2 \leqq \int \bar{x} \log(\bar{x}/x_n) \leqq \frac{1}{2} e^{E_n} E_n^2,$$

and hence the first result. The second part follows by using Proposition 2.3 in place of Lemma 2.2. □

Thus we see that for large n, if we can approximate $\log \bar{x}$ uniformly with a certain error by linear combinations of the a_i's, $i = 1, \cdots, n$, then x_n approximates $\bar{x}$ with error no worse (asymptotically) in the L_1 norm. In the last part of this section we shall see that when E_n is decreasing sufficiently quickly, as happens typically when $\bar{x}$ is sufficiently smooth, this actually forces uniform convergence of x_n to $\bar{x}$, due to the known form of x_n from Theorem 4.2.

We know that, by definition, $E_n \to 0$ exactly when $1 + \log \bar{x}$ lies in the closed span of the a_i's in L_∞. It follows from (5) that $\bar{x}$ is therefore uniquely determined by this fact and its moment sequence.

COROLLARY 4.8. *Suppose* $a_1 \equiv 1$, $\log x^1$ *and* $\log x^2$ *lie in* cl span $\{a_1, a_2, \cdots\}$, *and* $\int (x^1 - x^2) a_i = 0$, *for* $i = 1, 2, \cdots$. *Then* $x^1 = x^2$.

Proof. Clearly $0 \neq x^1$, $x^2 \in \operatorname{dom} I_\phi$, and without loss of generality we may assume that $\int x^1 = \int x^2 = 1$. If we set $\bar{x} := x^1$ in (P_n), then the corresponding sequence of optimal solutions $x_n^1 \to x^1$ in L_1, by (5). Similarly, setting $\bar{x} := x^2$ shows that the optimal solutions $x_n^2 \to x^2$. However, since x^1 and x^2 have identical moments, $x_n^1 = x_n^2$, for each n, so $x^1 = x^2$. □

DEFINITION 4.9. For each $n = 1, 2, \cdots$,

$$\Delta_n := \max \{\|f\|_\infty / \|f\|_1 \mid 0 \neq f \in \operatorname{span} \{a_1, \cdots, a_n\}\}.$$

We assume that at least one a_i is not identically zero, whence it is clear from positive homogeneity and compactness that Δ_n is well defined for large n with $0 < \Delta_n < +\infty$, and Δ_n is nondecreasing in n.

LEMMA 4.10. *For any* $f \in \operatorname{span} \{a_1, \cdots, a_n\}$,

$$\|f\|_\infty \leqq -\log (1 - \Delta_n \|e^f - 1\|_1),$$

where we interpret $\log (u) = -\infty$ *if* $u \leqq 0$.

Proof. We first claim that for any $u \geqq -M$, with $M > 0$,

$$|e^u - 1| \geqq M^{-1}(1 - e^{-M})|u|. \tag{7}$$

To see this, note that for $u \geqq 0$, $e^u - 1 \geqq u$, and $e^{-M} \geqq 1 - M$ by convexity, so $e^u - 1 \geqq u \geqq M^{-1}(1 - e^{-M})u$, as required. On the other hand, for $u \in [-M, 0]$, by convexity we have

$$\begin{aligned} \exp \{u\} &= \exp \{(-u/M)(-M) + (1 - (-u/M))0\} \\ &\leqq (-u/M) \exp (-M) + (1 - (-u/M)) \exp (0) \\ &= (-u/M)\, e^{-M} + 1 + (u/M), \end{aligned}$$

so we obtain $1 - e^u \geqq -M^{-1}(1 - e^{-M})u$, which gives (7).

Now to prove the lemma, we can suppose $f \neq 0$. Then from (7) we obtain, since $f \geqq -\|f\|_\infty$ almost everywhere, $|e^f - 1| \geqq \|f\|_\infty^{-1}(1 - e^{-\|f\|_\infty})|f|$ almost everywhere, so integrating shows

$$\|e^f - 1\|_1 \geqq \|f\|_\infty^{-1}(1 - e^{-\|f\|_\infty})\|f\|_1 \geqq \Delta_n^{-1}(1 - e^{-\|f\|_\infty})$$

by Definition 4.9. The result now follows. □

LEMMA 4.11. *With* E_n *defined as in Definition* 4.3, *there exists a* p_n *in* span $\{a_1, \cdots, a_n\}$, *satisfying* $\|p_n - 1 - \log \bar{x}\|_\infty = E_n$, *and hence*

$$\|e^{p_n - 1} - \bar{x}\|_1 \leqq E_n \left(1 + \frac{1}{2} E_n\, e^{E_n}\right) \int \bar{x}.$$

Proof. The first statement is just the definition of E_n. Now applying Lemma 4.4(b) gives $|e^{p_n - 1} - \bar{x}| \leqq E_n(1 + \frac{1}{2} E_n\, e^{E_n})\bar{x}$ almost everywhere, and integrating gives the result. □

We are now ready to obtain an estimate for the uniform error of the optimal solution x_n from $\bar{x}$.

THEOREM 4.12. *Suppose* $a_1 \equiv 1$, $\int \bar{x} = 1$, *and* $\log \bar{x} \in L_\infty$ (*or equivalently, for some* $\delta > 0$, $\delta \leqq \bar{x} \leqq \delta^{-1}$ *almost everywhere*). *Then with* E_n *and* Δ_n *defined in Definitions* 4.3 *and* 4.9, *the unique optimal solution of* (P_n),

$$x_n := e^{\sum_{i=1}^n \lambda_i^n a_i - 1},$$

has the property that

$$\|\log x_n - \log \bar{x}\|_\infty \leqq E_n - \log\{1 - (\operatorname{ess\,inf} \bar{x})^{-1} \Delta_n E_n e^{E_n}(1 + e^{E_n/2} + (E_n/2) e^{E_n})\}.$$

Proof. Since $\bar{x} > 0$ almost everywhere, (CQ) is satisfied, so x_n has the form given by Theorem 4.2, and

$$\|x_n - \bar{x}\|_1 \leqq E_n e^{E_n/2}, \tag{8}$$

by Theorem 4.7.

By Lemma 4.11, there exists p_n in span $\{a_1, \cdots, a_n\}$ with

$$\|p_n - 1 - \log \bar{x}\|_\infty = E_n, \tag{9}$$

$$\|e^{p_n - 1} - \bar{x}\|_1 \leqq E_n(1 + \tfrac{1}{2} E_n e^{E_n}). \tag{10}$$

If we write $q_n := \sum_{i=1}^n \lambda_i^n a_i$, we obtain from (8) and (10),

$$\begin{aligned} E_n(1 + \tfrac{1}{2} E_n e^{E_n} + e^{E_n/2}) &\geqq \|e^{q_n - 1} - e^{p_n - 1}\|_1 \\ &= \int e^{p_n - 1} |e^{q_n - p_n} - 1| \\ &\geqq \|e^{q_n - p_n} - 1\|_1 \operatorname{ess\,inf}(e^{p_n - 1}) \\ &\geqq \|e^{q_n - p_n} - 1\|_1 e^{-E_n} \operatorname{ess\,inf} \bar{x}, \end{aligned}$$

by (9). Thus we obtain

$$\|e^{q_n - p_n} - 1\|_1 \leqq E_n e^{E_n}(1 + \tfrac{1}{2} E_n e^{E_n} + e^{E_n/2})(\operatorname{ess\,inf} \bar{x})^{-1}.$$

We now apply Lemma 4.10, observing that $-\log(1 - \Delta_n u)$ is increasing in u:

$$\begin{aligned} \|q_n - p_n\|_\infty &\leqq -\log(1 - \Delta_n \|e^{q_n - p_n} - 1\|_1) \\ &\leqq -\log\{1 - (\operatorname{ess\,inf} \bar{x})^{-1} \Delta_n E_n e^{E_n}(1 + \tfrac{1}{2} E_n e^{E_n} + e^{E_n/2})\}. \end{aligned}$$

This, in combination with (9), gives the result. □

The error estimate in the above result shows that, providing, as n increases, that Δ_n is increasing at a slower rate than E_n is decreasing, then $\log x_n$ must converge to $\log \bar{x}$ in $\|\cdot\|_\infty$. More precisely, we have the following result.

COROLLARY 4.13. *Suppose* $a_1 \equiv 1$ *and* $\log \bar{x} \in L_\infty$. *Suppose further that* $\Delta_n E_n \to 0$ *as* $n \to \infty$, *where* E_n *and* Δ_n *are given by Definitions* 4.3 *and* 4.9, *respectively. Then the unique optimal solution of* (P_n), x_n, *converges in* $\|\cdot\|_\infty$ *to* $\bar{x}$ *as* $n \to \infty$. *In fact,* $\|x_n - \bar{x}\|_\infty = O(\Delta_n E_n)$, *as* $n \to \infty$.

Proof. Since $\bar{x} \neq 0$, we can, without loss of generality, scale μ so that $\int \bar{x}\, d\mu = 1$. Since Δ_n is nondecreasing in n, we have that $E_n \to 0$, so Theorem 4.12 shows that

$$\begin{aligned} \overline{\lim_{n \to \infty}} \|\log x_n - \log \bar{x}\|_\infty / \Delta_n E_n &\leqq \left(\lim_{n \to \infty} \Delta_n^{-1} \right) + 2(\operatorname{ess\,inf} \bar{x})^{-1} \\ &\leqq \mu(S) + 2(\operatorname{ess\,inf} \bar{x})^{-1}, \end{aligned}$$

since $\Delta_n \geqq \Delta_1 = \mu(S)^{-1}$. Thus for some constant k (independent of n), $|\log x_n - \log \bar{x}| \leqq k\Delta_n E_n$ almost everywhere for all n; so by Lemma 4.4(b),

$$|x_n - \bar{x}| \leqq (k\Delta_n E_n)(1 + e^{(k\Delta_n E_n)}(k\Delta_n E_n)/2)\bar{x} \quad \text{a.e.}$$

Thus $\overline{\lim}_{n\to\infty} \|x_n - \bar{x}\|_\infty / \Delta_n E_n \leqq k\|\bar{x}\|_\infty$, and the result follows. □

So we see from the above that if $E_n = O(n^{-\alpha})$ and $\Delta_n = O(n^\beta)$, where $\beta < \alpha$, then x_n converges to $\bar{x}$ in $\|\cdot\|_\infty$ with error no larger than $O(n^{\beta-\alpha})$. In general, E_n (and hence α) will depend on the smoothness of $\bar{x}$, whereas Δ_n (and β) depends only on the constraint functions $\{a_1, \cdots, a_n\}$.

Once we know that $\|x_n - \bar{x}\|_\infty \to 0$, we can replace the use of (5) in Theorem 4.7 by (6). Following through the above argument, and replacing $\|\cdot\|_1$ by $\|\cdot\|_p$ where appropriate ($p \geqq 2$), gives the slightly refined estimate $\|x_n - \bar{x}\|_\infty = O(\Delta_{n,p} E_n^{2/p})$, where

$$\Delta_{n,p} := \max\{\|f\|_\infty / \|f\|_p \mid 0 \neq f \in \operatorname{span}\{a_1, \cdots, a_n\}\}. \tag{11}$$

In particular,

$$\|x_n - \bar{x}\|_\infty = O(\Delta_{n,2} E_n). \tag{12}$$

In the final section we consider two classical cases where explicit bounds are known for E_n and Δ_n. This allows us to show that for algebraic and trigonometric moment problems on intervals, if the underlying density $\bar{x}$ is sufficiently smooth and strictly positive, the estimates x_n converge uniformly to $\bar{x}$.

5. The classical algebraic and trigonometric moment problems. We begin by summarizing Corollary 4.13. We consider the problem

$$(P_n) \qquad \begin{cases} \text{minimize} & I_\phi(x) \\ \text{subject to} & \displaystyle\int_S (x - \bar{x}) a_i \, d\mu = 0, \qquad i = 1, \cdots, n, \\ & 0 \leqq x \in L_1(S, \mu), \end{cases}$$

where we suppose $\log \bar{x} \in L_\infty$, and $a_1 \equiv 1$, and we denote the unique optimal solution by x_n. Then Corollary 4.13 states that $\|x_n - \bar{x}\|_\infty \to 0$, providing $E_n \Delta_n \to 0$, where E_n and Δ_n are given by Definitions 4.3 and 4.9, respectively:

$$E_n = \min\{\|f - \log \bar{x}\|_\infty \mid f \in \operatorname{span}\{a_1, \cdots, a_n\}\},$$

$$\Delta_n = \max\{\|f\|_\infty / \|f\|_1 \mid 0 \neq f \in \operatorname{span}\{a_1, \cdots, a_n\}\}.$$

We consider two special cases.

Algebraic moment problems. In this case, $S = [0, 1]$, μ is Lebesgue measure, and $a_i(s) = s^{i-1}$, for $i = 1, \cdots, n$.

THEOREM 5.1. *Suppose, for the algebraic moment problem, that $\bar{x}$ is twice continuously differentiable and strictly positive. Then $n^2 E_n \to 0$ as $n \to \infty$.*

Proof. Since $0 < \bar{x} \in C^2[0, 1]$, it follows that $\log \bar{x} \in C^2[0, 1]$, so by Jackson's theorem [9], for some constant k, $E_n \leqq (k/n^2)\omega((\log \bar{x})'', 1/n)$, where

$$\omega(g, \delta) := \sup\{|g(s) - g(t)| \mid |s - t| \leqq \delta, s, t \in [0, 1]\},$$

is the modulus of continuity. Since $\omega(g, 0+) = 0$ for continuous g, the result follows. □

THEOREM 5.2. *For the algebraic moment problem,*

$$n^2 \geqq \Delta_n \geqq \begin{cases} (n+1)^2/4, & n \text{ odd}, \\ n(n+2)/4, & n \text{ even}. \end{cases}$$

□

Proof. For the proof, see [1].

COROLLARY 5.3. *Suppose, for the algebraic moment problem, that $\bar{x}$ is twice continuously differentiable and strictly positive. Then the unique optimal solutions x_n converge uniformly to $\bar{x}$.*

Proof. For the proof, see Corollary 4.13 and Theorems 5.1 and 5.2. □

In fact, a rather more precise version of the above argument, using (12), shows that if $\bar{x}$ is k times continuously differentiable ($k \geqq 2$) and strictly positive, then $\|x_n - \bar{x}\|_\infty = o(n^{1-k})$: the relevant Jackson theorem states that in this case $E_n = o(n^{-k})$, while it is shown in [1] that $\Delta_{n,2} = n$. We also see in this case, from Theorem 4.7, that for any $2 \leqq p < +\infty$, $\|x_n - \bar{x}\|_p = o(n^{-2k/p})$. In particular, $\|x_n - \bar{x}\|_2 = o(n^{-k})$.

In general, the smoother $\bar{x}$ is, the more rapidly E_n tends to zero. If $\bar{x}$ is analytic on [0, 1], or in other words has an analytic extension to an open subset of the complex plane containing [0, 1], then $E_n \to 0$ linearly: $E_n = O(\rho^n)$ for some constant $0 \leqq \rho < 1$. If, in fact, $\bar{x}$ is an entire function, the convergence is superlinear (see [16]). It follows from Corollary 4.13 that for analytic, strictly positive $\bar{x}$ in the algebraic moment problem, $\|x_n - \bar{x}\|_\infty \to 0$ linearly with the same convergence ratio as E_n; and if $\bar{x}$ is entire, the convergence is superlinear.

Trigonometric moment problems. In this case, $S = [-\pi, \pi]$, $2\pi\mu$ is Lebesgue measure, and for $j = 1, 2, \cdots$, $a_{2j}(s) = \cos(js)$ and $a_{2j+1}(s) = \sin(js)$.

THEOREM 5.4. *Suppose, for the trigonometric moment problem, that $\bar{x}$ is strictly positive with both $\bar{x}$ and $\bar{x}'$ continuous and 2π-periodic. Then, $nE_{2n+1} \to 0$ as $n \to \infty$.*

Proof. By [9], $E_{2n+1} \leqq (578/n)\omega((\log \bar{x})', 1/n)$, where again $\omega(\cdot, \cdot)$ is the modulus of continuity; so the result follows. □

THEOREM 5.5. *For the trigonometric moment problem, $2n+1 \geqq \Delta_{2n+1} \geqq n$.*

Proof. For the proof, see [22]. □

COROLLARY 5.6. *Suppose, for the trigonometric moment problem, that $\bar{x}$ is strictly positive with both $\bar{x}$ and $\bar{x}'$ continuous and 2π-periodic. Then the unique optimal solutions x_n converge uniformly to $\bar{x}$.*

Proof. Theorems 5.4 and 5.5 show that

$$\Delta_{2n+1}E_{2n+1} \leqq (2n+1)E_{2n+1} \to 0,$$

$$\Delta_{2n+2}E_{2n+2} \leqq \Delta_{2n+3}E_{2n+1} \leqq (2n+3)E_{2n+1} \to 0.$$

Thus $\Delta_n E_n \to 0$, so the result follows by Corollary 4.13. □

In fact, just as in the algebraic case, a more precise version of the above argument (using the fact that $\Delta_{2n+1,2} = (2n+1)^{1/2}$ in this case [22]) shows that if $\bar{x}, \bar{x}', \cdots, \bar{x}^{(k)}$ are continuous and 2π-periodic, with $\bar{x}$ strictly positive, then $\|x_n - \bar{x}\|_\infty = o(n^{(1/2)-k})$. Furthermore, Theorem 4.7 shows that for any $2 \leqq p < +\infty$, $\|x_n - \bar{x}\|_p = o(n^{-2k/p})$. In particular, $\|x_n - \bar{x}\|_2 = o(n^{-k})$.

Our approach can be extended to prove similar results for multidimensional algebraic and trigonometric moment problems. Thus one can consider polynomials with maximum degree or sum of degrees not exceeding n, etc., on various domains. This becomes considerably more technical and we choose not to take the matter further herein.

Note added in proof. Error bounds for the trigonometric case under certain conditions on $\bar{x}$ (and numerical results) may be found in [23], and bounds for problems involving some entropies other than the Boltzmann-Shannon entropy appear in [24].

REFERENCES

[1] D. AMIR AND Z. ZIEGLER, *Polynomials of extremal L_p-norm on the L_∞-unit sphere*, J. Approx. Theory, 18 (1976), pp. 86–98.

[2] J. M. BORWEIN AND A. S. LEWIS, *Duality relationships for entropy-like minimization problems*, SIAM J. Control Optim., 29 (1991), pp. 325–338.

[3] ———, *On the convergence of moment problems*, Trans. Amer. Math. Soc., 1991, to appear.

[4] J. M. BORWEIN, A. S. LEWIS, AND R. NUSSBAUM, *Entropy minimization, DAD problems and doubly-stochastic kernels*, to appear.

[5] W. BRITTON, *Conjugate duality and the exponential Fourier spectrum*, Lecture Notes in Statistics 18, Springer-Verlag, New York, 1983.

[6] I. CSISZÁR, *Information-type measures of difference of probability distributions and indirect observations*, Studia Sci. Math. Hungar., 2 (1967), pp. 299–318.

[7] A. DECARREAU, D. HILHORST, C. LEMARÉCHAL, AND J. NAVAZA, *Dual methods in entropy maximization: Application to some problems in crystallography*, SIAM J. Optimization, submitted.

[8] J. DIESTEL, *Sequences and Series in Banach Spaces*, Springer-Verlag, New York, 1984.

[9] R. P. FEINERMAN AND D. J. NEWMAN, *Polynomial Approximation*, Williams and Wilkins, Baltimore, MD, 1974.

[10] B. FORTE, W. HUGHES, AND Z. PALES, Maximum entropy estimators and the problem of moments, Rend. Mat. Ser. VII, 9 (1989), pp. 689–699.

[11] S. M. KAY AND S. L. MARPLE, *Spectrum analysis—a modern perspective*. Proc. IEE-F, 69 (1981), pp. 1380–1419.

[12] J. H. B. KEMPERMAN, *On the optimum rate of transmitting information*, in Probability and Information Theory, Proceedings of an International Symposium, McMaster University, Hamilton, Ontario, pp. 126–169; Lecture Notes in Mathematics 89, Springer-Verlag, Berlin, 1969.

[13] S. KULLBACK, A lower bound for discrimination information in terms of variation, IEEE Trans. Inform. Theory, IT-13 (1967), pp. 126–127.

[14] S. W. LANG AND J. H. MCCLELLAN, *Spectral estimation for sensor arrays*, IEEE Trans. Acoust. Speech Signal Process., ASSP-31 (1983), pp. 349–358.

[15] A. S. LEWIS, *The convergence of entropic estimates for moment problems*, in Workshop/Miniconference on Functional Analysis/Optimization, S. Fitzpatrick and J. Giles, eds., Centre for Mathematical Analysis, Australian National University, Canberra, Australia, 1988, pp. 100–115.

[16] G. G. LORENTZ, *Approximation of Functions*, Second Edition, Chelsea, New York, 1986.

[17] L. R. MEAD AND N. PAPANICOLAOU, *Maximum entropy in the problem of moments*, J. Math. Phys., 25 (1984), pp. 2404–2417.

[18] R. T. ROCKAFELLAR, *Integrals which are convex functionals*, Pacific J. Math., 24 (1968), pp. 525–539.

[19] ———, *Integrals which are convex functionals*, II, Pacific J. Math., 39 (1971), pp. 439–469.

[20] J. SKILLING AND S. F. GULL, *The entropy of an image*, SIAM-AMS Proc., Applied Mathematics, 14 (1984), pp. 167–189.

[21] K. R. STROMBERG, *An Introduction to Classical Real Analysis*, Wadsworth, Belmont, CA, 1981.

[22] Z. ZIEGLER, *Minimizing the $L_{p,\infty}$-distortion of trigonometric polynomials.*, J. Math. Anal. Appl., 61 (1977), pp. 426–431.

[23] E. GASSIAT, *Problème sommatoire par maximum d'entropie*, C. R. Acad. Sci. Paris Sér. I, 303 (1986), pp. 675–680.

[24] D. DACUNHA-CASTELLE AND F. GAMBOA, *Maximum d'entropie et problème des moments*, Ann. Inst. H. Poincaré Probab. Statist., 26 (1990), pp. 567–596.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 206-221, May 1991

004

AN EFFICIENT IMPLEMENTATION OF MERRILL'S METHOD FOR SPARSE OR PARTIALLY SEPARABLE SYSTEMS OF NONLINEAR EQUATIONS*

YANG BING† AND GAO LIN‡

Abstract. When Merrill's method without an extra dimension is applied to solving sparse or partially separable systems of nonlinear equations, the computational efficiency can be further improved, i.e., a larger piece of linearity can be traversed in one step by using a suitable linear system. One of the linear systems is updated and the corresponding technique is shown to update information about the linear system and the large piece in the implementation of the method. Some numerical results of the method support the claim that it is efficient. Also, some mistakes from a previous paper, in which the main technique exploiting sparsity was proposed, are corrected.

Key words. fixed point algorithm, systems of nonlinear equations, variable dimension algorithm, sparsity, partial separability

AMS(MOS) subject classification. 1T90

1. Introduction. Fixed point algorithms, initiated by Scarf [15], have been applied to systems of nonlinear equations that come from economic equilibrium theory, game theory, nonlinear programming, operations research, and various fields of engineering. The common character of these algorithms is their global convergence under suitable conditions, which is not possessed by traditional analytic methods, such as Newton's method. In general, however, fixed point algorithms are less efficient than analytic methods. Many scholars have studied how to improve the efficiency of the fixed point algorithm (see, e.g., [1]). One of their research results is the variable dimension algorithm. Kojima, Yamamoto, Todd, Freund, and others have developed this algorithm ([2], [4], [5], [9]), and its prototype was proposed by van der Laan and Talman [6]. Another of the results is the technique to improve computational efficiency by exploiting structure for a system of nonlinear equations which has special structures, such as partial linearity, separability, partial separability, or sparsity. Kojima, Todd, and others have proposed this technique ([3], [10], [11], [12], [13]). We can expect that a variable dimension algorithm combined with a technique to improve computational efficiency by exploiting structure is more efficient for systems of nonlinear equations which have special structures. Such systems often appear in models of practical problems.

Todd developed a pivot-saving technique for sparse systems of nonlinear equations in [13]. Kojima and Yamamoto pointed out that the technique can be applied to the variable dimension algorithms that they developed [5]. In this paper, our contributions are as follows:

(1) We point out that the technique in [13] can also be exploited for partially separable systems, including separable systems, of nonlinear equations. We practically apply the technique to Merrill's method using Kojima and Yamamoto's PDM structure

* Received by the editors March 2, 1990; accepted for publication (in revised form) November 26, 1990. This research was supported by the National Natural Science Foundation of China.

† Department of Management Engineering, Harbin Shipbuilding Engineering Institute, Harbin 150001, China.

‡ Department of Mathematics and Mechanics, Harbin Shipbuilding Engineering Institute, Harbin 15001, China.

[14] and obtain a variable dimension algorithm for sparse or partially separable systems of nonlinear equations. In this algorithm a large piece of linearity can be traversed in one step by using a suitable linear system and the corresponding technique to update information about a large piece. Because we use a different homotopy, the PDM structure, and other treatments, our linear system and its updating technique are different from those in [13].

(2) We present some numerical results of our algorithm and prove that it is efficient.

(3) From our computational experience, we suspect that some details of the updating technique in [13] are mistaken, thus we will point them out and show how to revise them.

In § 2, we first introduce the pivot-saving technique for sparse systems of nonlinear equations given by Todd [13] in order to make our work understood. Then, we give the definition of partial separability and point out that the technique in [13] can also be used for partially separable systems, including separable systems, of nonlinear equations. In § 3, we present the linear system and the expression of a large piece of linearity in our method. In § 4, we show how to update the linear system corresponding to one large piece in order to obtain the appropriate system for an adjacent large piece and to revise some details of the updating technique in [13]. In § 5, we report some numerical results which prove that the implementation of the method and the kind of pivot-saving technique proposed in [13] are efficient.

2. The pivot-saving technique. In Todd [13] a pivot-saving technique for sparse systems of nonlinear equations was proposed. This technique is the main basis of the present paper. First, let us introduce its outline.

When piecewise-linear homotopy algorithms are applied to the problem of approximating a zero of a sparse function $f: R^n \to R^n$, a large piece of linearity can be traversed in one step by using a suitable linear system.

Confining ourselves to Merrill's method, we define $h: R^n \times [0,1] \to R^n$ by $h(x,t) = tf(x) + (1-t)f^0(x)$, where $f^0: R^n \to R^n$ is a simple function, and we take $f^0(x) = M(x - x^0)$, where matrix M is $n \times n$ and nonsingular. Let 1 be the piecewise-linear approximation to h with respect to the triangulation $\tilde{J}_1 = \{j_1(v, \pi, s)\}$ of $R^n \times [0,1]$, where $v \in R^n \times \{1\}$ has all v_i's odd, π is a permutation of $\{1, 2, \cdots, n+1\}$ with $\pi(j) = n+1$, and $s \in R^n \times \{-1\}$ is a sign vector; each s_i is ± 1. The algorithm traces a path of zeros of 1 starting from $(x^0, 0)$. If it reaches some point $(x^1, 1)$, then x^1 is an approximate zero of f. We then either stop, or restart the algorithm with a smaller grid size, x^1, replacing x^0, and possibly a new matrix M. See, for example, [1]. Let e^p denote the pth unit vector of appropriate dimension. Then the vertices of σ with the grid size ε are $v^0, v^1, \cdots, v^{n+1}$, where

$$v^0 = v, \qquad v^i = v^{i-1} + \varepsilon s_{\pi(i)} e^{\pi(i)}, \qquad 1 \leqq i < j,$$

$$v^j = v^{j-1} - e^{n+1}, \qquad v^k = v^{k-1} + \varepsilon s_{\pi(k)} e^{\pi(k)}, \qquad j < k \leqq n+1.$$

Because f^0 is affine, the pieces of linearity of 1 are larger than the simplices of $\tilde{J}_1$. These pieces form a (polyhedral) subdivision $\hat{J}_1 = \{\hat{j}_1(v, \pi, s)\}$ of $R^n \times [0,1]$ (see [10]). The individual piece $\hat{\sigma} \equiv \hat{j}_1(v, \pi, s)$, with $\pi(j) = n+1$, is the set of all $w \in R^{n+1}$ satisfying

$$\varepsilon \geqq s_{\pi(1)}(w_{\pi(1)} - v_{\pi(1)}) \geqq \cdots \geqq s_{\pi(j-1)}(w_{\pi(j-1)} - v_{\pi(j-1)}) \geqq \varepsilon(1 - w_{n+1});$$

$$\varepsilon(1 - w_{n+1}) \geqq |w_{\pi(k)} - v_{\pi(k)}|, \qquad j < k \leqq n+1;$$

$$\varepsilon(1 - w_{n+1}) \geqq 0 \quad \text{if } j = n+1.$$

Here an inequality corresponds to a facet of $\hat{\sigma}$. Introducing some notation, we can

rewrite these inequalities as $C_{\hat{\sigma}} w \geqq d^{\hat{\sigma}}$, where each row of matrix $C_{\hat{\sigma}}$ corresponds to a facet of $\hat{\sigma}$. Let $A_{\hat{\sigma}}$ denote the derivative matrix of the affine function from R^{n+1} to R^n that agrees with 1 on $\hat{\sigma}$ and let a^i denote the ith column of $A_{\hat{\sigma}}$; we have

$$a^{\pi(i)} = (f(y^i) - f(y^{i-1}))/\varepsilon s_{\pi(i)}, \qquad 1 \leqq i < j;$$

$$a^{\pi(k)} = m^{\pi(k)}, \qquad j < k \leqq n+1;$$

$$a^{n+1} = f(y^j) + f^0(y^j) \quad \text{and} \quad 1(w) = 1(\bar{w}) + A_{\hat{\sigma}}(w - \bar{w}) \quad \text{for any } w, \bar{w} \in \hat{\sigma},$$

where y^i is the projection of vertex $v^i \in R^n \times [0, 1]$ on R^n and m^k is the kth column of the matrix M. Suppose f is sparse (for example, there is no component of f that depends on both x_p and x_q). If $\hat{\pi}(i) = p$ and $\hat{\pi}(i+1) = q$, $1 \leqq i \leqq j-1$, and $\tilde{\pi}$ denotes π with the positions of p and q interchanged, then we find $A_{\hat{\sigma}} = A_{\tilde{\sigma}}$ with $\hat{\sigma} = \hat{j}_1(v, \hat{\pi}, s)$ and $\tilde{\sigma} = \hat{j}_1(v, \tilde{\pi}, s)$. It follows that 1 is linear in $\check{\sigma} = \hat{\sigma} \cup \tilde{\sigma}$. Then we can say that when f is sparse, a large piece of linearity of the homotopy 1, $\check{\sigma}$, meets $1^{-1}(0)$ in a line segment; this line segment can be traversed by considering a linear system $Aw = b$, $Cw \geqq d$. Here $A = A_{\hat{\sigma}}$, $b = A\bar{w} - 1(\bar{w})$, with $\bar{w}$ any point in $\check{\sigma}$, and a row of the matrix C corresponds to a facet of the piece $\check{\sigma}$, the number of rows being equal to the number of facets. We trace the segment numerically by generating a particular solution $\bar{w}$ to $Aw = b$, $Cw \geqq d$ (corresponding to where the piece is entered) and a vector z in the null space of A. Then $\{w: Aw = b\} = \{\bar{w} + \lambda z\}$. After that we find where the segment leaves the piece by finding the range of λ for which $C(\bar{w} + \lambda z) \geqq d$; this corresponds to a minimum ratio test in linear programming.

Some details of an implementation of a piecewise-linear homotopy algorithm using these large pieces are given in Todd [13]. In particular, Todd describes what must be stored and how to update the matrix A, and he demonstrates the directed graph Δ which is used to identify the large piece $\check{\sigma}$.

Now we introduce the partial separability of f. If we write $x \in R^n$ as $x = (x_p, x_q, \tilde{x})$, $\tilde{x} \in R^{n-2}$, and there are functions $f^p: R \times R^{n-2} \to R^n$ and $f^q: R \times R^{n-2} \to R^n$ such that $f(x) = f^p(x_p, \tilde{x}) + f^q(x_q, \tilde{x})$, then we say that coordinates p and q are separable. If f is differentiable, p and q are separable if the pth column of the derivative matrix $Df(x)$ does not depend on x_q and the qth column of $Df(x)$ does not depend on x_p. If all pairs (p, q) for $1 \leqq p, q \leqq n$, and $p \neq q$ are separable, then the function f is separable. If there is at least one separable pair (p, q), the f is partially separable. Clearly, $A_{\hat{\sigma}} = A_{\tilde{\sigma}}$ when $p = \hat{\pi}(i)$ and $q = \hat{\pi}(i+1)$, $1 \leqq i \leqq j-1$, are separable. We say the pivot-saving technique in [13] can be fully applied to separable or partially separable systems of nonlinear equations.

3. The linear system and the large piece.

3.1. The homotopy. Suppose we solve the system of nonlinear equations

$$f(x) = 0 \qquad (x \in R^n) \tag{3.1}$$

where R^n is an n-dimensional Euclidean space and f is a continuous mapping from R^n into itself. When a rough approximation of a solution of the system (3.1), w^0, is known, we let

$$g(x) = f(w^0 + x) \quad (\text{for every } x \in R^n). \tag{3.2}$$

We may deal with the system of equations

$$g(x) = 0 \qquad (x \in R^n) \tag{3.3}$$

instead. We define the mapping

$$h(x, y, t) = My + tg(x) \qquad ((x, y, t) \in R^n \times R^n \times R_+), \tag{3.4}$$

with M a (sparse when g is sparse, with the same pattern) nonsingular matrix "close" to the Jacobian of g, and we solve the system

$$h(x, y, t) = 0, \qquad ((x, y, t) \in R^n \times R^n \times R_+). \tag{3.5}$$

Under certain assumptions, the connected component of $h^{-1}(0)$ containing the point $(0, 0, 0)$ is a smooth path converging to $(\bar{x}, \bar{y}, +\infty)$. Then

$$g(\bar{x}) = 0, \tag{3.6}$$

so that $\bar{x}$ is a solution of (3.3) and $w^0 + \bar{x}$ is a solution of (3.1).

3.2. The PDM. The primal-dual pair of subdivided manifolds (abbreviated as PDM), given by Kojima and Yamamoto [4] for an efficient implementation of Merrill's method, is utilized here. In the case of $n = 2$, the subdivided primal and dual both look like a "checkerboard," see Fig. 3.1.

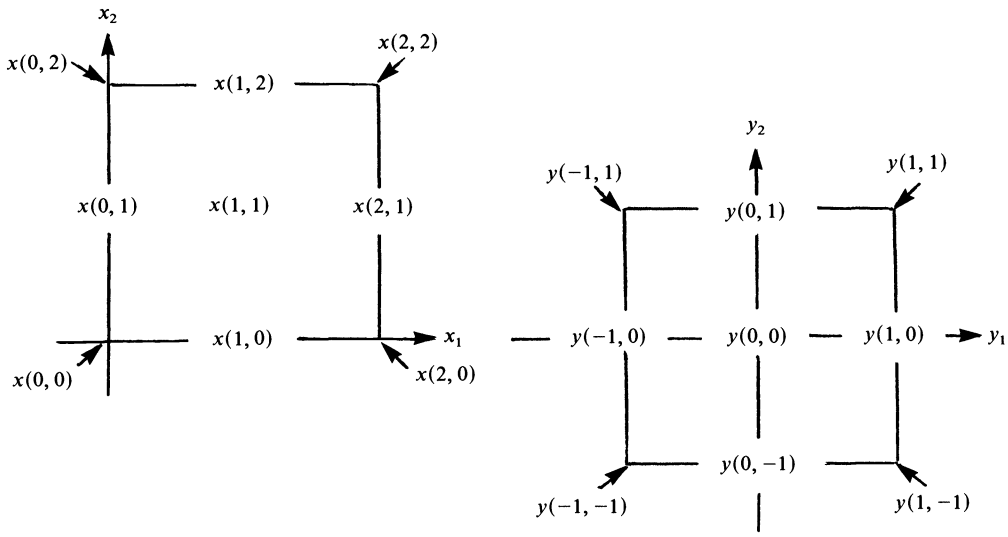


FIG. 3.1

Let ε be an arbitrarily fixed positive number and let Q be the set of all n-dimensional vectors $q = (q_1, q_2, \cdots, q_n)$ such that each q is an integral multiple of ε. For every $q \in Q$, let

$$\begin{aligned}
&I_e(q) = \{i\colon q_i \text{ is an even multiple of } \varepsilon\},\\
&I_0(q) = \{j\colon q_j \text{ is an odd multiple of } \varepsilon\},\\
&X(q) = \{x \in R^n\colon x_i = q_i \text{ for } i \in I_e(q),\ q_j - \varepsilon \leqq x_j \leqq q_j + \varepsilon \text{ for } j \in I_0(q)\},\\
&Y(q) = \{y \in R^n\colon q_i - \varepsilon \leqq y_i \leqq q_i + \varepsilon \text{ for } i \in I_e(q),\ y_j = q_j \text{ for } j \in I_0(q)\};
\end{aligned} \tag{3.7}$$

then

$$\begin{aligned}
P &= \{X(q)\colon q \in Q \text{ and } I_e(q) = \varnothing\},\\
D &= \{Y(q)\colon q \in Q \text{ and } I_0(q) = \varnothing\}
\end{aligned}$$

are subdivisions of R^n. The dual operator d is defined as

$$X^d(q) = Y(q), \quad Y^d(q) = X(q) \qquad (\text{for every } q \in Q). \tag{3.8}$$

Let T be a simplicial refinement of the primal subdivided manifold P. For every cell X of $\bar{P}$ ($\bar{P}=\{X(q): q\in Q\}$), let

$$T|X=\{\sigma\in T: \sigma\subset X, \dim\sigma=\dim X\}.$$

Define

$$N=\{\sigma\times X^d\times R_+: \sigma\in T|X\in\bar{P}\} \tag{3.9}$$

and

$$\mu=\{X\times X^d\times R_+: X\in\bar{P}\}; \tag{3.10}$$

then N is a refinement of the subdivided $(n+1)$-manifold μ.

3.3. The linear system and the large piece of linearity. Letting $G: R^n\to R^n$ be a simplicial approximation of g, we replace the mapping of (3.4) by the mapping $H: R^n\times R^n\times R_+\to R^n$:

$$H(x, y, t)=My+tG(x)\qquad((x, y, t)\in R^n\times R^n\times R_+). \tag{3.11}$$

By tracing a path in the solution set $H^{-1}(0)$ of the system (3.11), we can obtain an approximate solution to the system (3.3).

A modification of the triangulation J_1 of R^n is used in the simplicial refinement of P. A k-dimensional simplex σ is

$$\begin{aligned} &x^0: x_i^0 \quad \text{is an even multiple of } \varepsilon && \text{for } 1\leqq i\leqq n,\\ &x^i=x^{i-1}+\varepsilon s_{\pi(i)}\, e^{\pi(i)} && \text{for } 1\leqq i\leqq k,\\ &\sigma=co(x^0, x^1, \cdots, x^k). \end{aligned} \tag{3.12}$$

Let $z=\sigma\times X^d(q)\times R_+$ with $q=x^k$, $\dim\sigma=k(0\leqq k\leqq n)$, $H^{-1}(0)\cap z\neq\varnothing$. Then

$$\begin{aligned} &I_0(q)=\{\pi(1), \pi(2), \cdots, \pi(k)\},\\ &I_e(q)=\{1, 2, \cdots, n\}\backslash I_0(q), \end{aligned} \tag{3.13}$$

and σ is the set of all x satisfying

$$\begin{aligned} &\varepsilon\geqq s_{\pi(1)}(x_{\pi(1)}-x^0_{\pi(1)})\geqq\cdots\geqq s_{\pi(k)}(x_{\pi(k)}-x^0_{\pi(k)})\geqq 0\\ &x_i=q_i \quad (\text{for all } i\in I_e(q)). \end{aligned} \tag{3.14}$$

As

$$G(x)=(a^{\pi(1)}a^{\pi(2)}\cdots a^{\pi(k)})\begin{bmatrix} x_{\pi(1)}-x^0_{\pi(1)}\\ x_{\pi(2)}-x^0_{\pi(2)}\\ \vdots\\ x_{\pi(k)}-x^0_{\pi(k)}\end{bmatrix}+g(x^0)\quad(\text{for all } x\in\sigma), \tag{3.15}$$

where $a^{\pi(i)}=(g(x^i)-g(x^{i-1}))/\varepsilon s_{\pi(i)}$ (for $1\leqq i\leqq k$), we have

$$\begin{aligned} My+tG(x)=My&+(a^{\pi(1)}a^{\pi(2)}\cdots a^{\pi(k)})\begin{bmatrix} tx_{\pi(1)}\\ tx_{\pi(2)}\\ \vdots\\ tx_{\pi(k)}\end{bmatrix}\\ &+\left(g(x^0)-(a^{\pi(1)}a^{\pi(2)}\cdots a^{\pi(k)})\begin{bmatrix} x^0_{\pi(1)}\\ x^0_{\pi(2)}\\ \vdots\\ x^0_{\pi(k)}\end{bmatrix}\right)t\quad(\text{for all } (x, y, t)\in z). \end{aligned} \tag{3.16}$$

Let $u = tx$, $v = y$, and $w_j = u_j$ $(j \in I_o(q))$, $w_i = v_i$ $(i \in I_e(q))$, and $w_{n+1} = t$. Thus the system (3.5) in z becomes the linear system

$$Aw = b \tag{3.17}$$

where

$$\begin{aligned} A\colon\ & a^{\pi(i)} = (g(x^i) - g(x^{i-1}))/\varepsilon s_{\pi(i)} \qquad (\pi(i) \in I_o(q)) \\ & a^i = m^i \qquad (i \in I_e(q)) \end{aligned}$$

$$a^{n+1} = g(x^0) - (a^{\pi(1)} a^{\pi(2)} \cdots a^{\pi(k)}) \begin{bmatrix} x^0_{\pi(1)} \\ x^0_{\pi(2)} \\ \vdots \\ x^0_{\pi(k)} \end{bmatrix}, \tag{3.18}$$

$$b = -\sum_{i=1}^{k} q_{\pi(i)} m^{\pi(i)} \qquad (\pi(i) \in I_0(q)),$$

and $(x, y, t) \in z$ if and only if (u, v, t) satisfies the linear system

$$\begin{aligned} & \varepsilon t \geqq s_{\pi(1)}(u_{\pi(1)} - x^0_{\pi(1)} t) \geqq \cdots \geqq s_{\pi(k)}(u_{\pi(k)} - x^0_{\pi(k)} t) \geqq 0 \\ & u_i = q_i t \qquad (i \in I_e(q)) \\ & -\varepsilon \leqq v_i - q_i \leqq \varepsilon \qquad (i \in I_e(q)) \\ & v_j = q_j \qquad (j \in I_0(q)) \\ & t \geqq 0. \end{aligned} \tag{3.19}$$

In order to describe concisely the piece of linearity, we use some notation similar to that used in [13]. Let

$$\begin{aligned} & \varepsilon t = \gamma^0 w - \delta_0, \qquad s_j(u_j - x^0_j t) = \gamma^j w - \delta_j \qquad (j \in I_0(q)), \\ & v_i - q_i = \gamma^i w - \delta_i \qquad (i \in I_e(q)), \\ & 0 = \gamma^{n+1} w - \delta_{n+1}, \qquad +\varepsilon = \gamma^{n+2} w - \delta_{n+2}, \qquad -\varepsilon = \gamma^{n+3} w - \delta_{n+3}, \end{aligned}$$

and

$$c^{pp'} = \gamma^p - \gamma^{p'}, \qquad d_{pp'} = \delta_p - \delta_{p'} \qquad (p, p' \in N_+ \equiv \{0, 1, \cdots, n+1, n+2, n+3\}).$$

Let graph Γ consist of the edges

$$(0, \pi(1)), (\pi(1), \pi(2)), \cdots, (\pi(k), n+1)$$

and

$$(n+2, i), (i, n+3) \quad (\text{for all } i \in I_e(q)).$$

Then the inequalities of (3.19) are of the form:

$$\begin{aligned} & c^{pp'} w \geqq d_{pp'} \qquad ((p, p') \in \Gamma) \\ & w_{n+1} \geqq 0 \end{aligned}$$

or

$$C(\Gamma) w \geqq d(\Gamma), \qquad w_{n+1} \geqq 0. \tag{3.20}$$

This is the inequality system corresponding to $\sigma \times X^d(q) \times R_+$.

We now have Theorem 1.

THEOREM 1. *The set of* (x, y, t) *lying in* $H^{-1}(0) \cap z$ *is the set of solutions to* $Aw = b$, $C(\Gamma)w \geqq d(\Gamma)$, *and* $w_{n+1} \geqq 0$. *That is, if* w *is a solution to the latter, then* (x, y, t), *given by*

$$
\begin{aligned}
x_j &= \begin{pmatrix} w_j / w_{n+1} & (w_{n+1} > 0) \\ 0 & (w_{n+1} = 0) \end{pmatrix} \quad (j \in I_o(q)), \\
x_i &= q_i \qquad (i \in I_e(q)), \\
y_i &= w_i \qquad (i \in I_e(q)), \\
y_j &= q_j \qquad (j \in I_o(q)), \\
t &= w_{n+1},
\end{aligned} \tag{3.21}
$$

lies in $H^{-1}(0) \cap z$, *where* A *and* b *are given by* (3.18).

Suppose a function f is sparse or partially separable. Let $\check{z} = \check{\sigma} \times X^d(q) \times R_+$, $\check{\sigma} \subset X(q)$, and $\check{z}$ be a large piece of linearity for (3.17). We can define the graph Δ as in [13], so that each pair (directed edge) $(p, p') \in \Delta$ $(p, p' \in I_o(q))$ corresponds to a facet of $\check{\sigma}$.

We have the following theorem.

THEOREM 2. *The set of* (x, y, t) *lying in* $H^{-1}(0) \cap \check{z}$ *is the set of solutions to* $Aw = b$, $C(\Delta)w \geqq d(\Delta)$, *and* $w_{n+1} \geqq 0$. *That is, if* w *is a solution to the latter, the point* (x, y, t) *satisfying the system* (3.21) *lies in* $H^{-1}(0) \cap \check{z}$, *where* A *and* b *are given by* (3.18), *and the vector* π *corresponds to some simplex* σ *of* $\check{\sigma}$.

4. Implementation.

4.1. Iterative steps. Suppose a point $(\bar{x}, \bar{y}, \bar{t})$, with $H(\bar{x}, \bar{y}, \bar{t}) = 0$ where the piece $\check{z}$ is entered, is given and we have a corresponding $\bar{w}$ with $A\bar{w} = b$. Let $A = (B\, a^{n+1})$, and assume for simplicity that B is nonsingular, although the algorithm might generate a matrix A (of rank n) with B singular. An LU factorization of B (or in fact of A) is preferable, not only for maintaining sparsity, but also for numerical stability [12]. However, we use the explicit inverse of B for simplicity. Then

$$
\delta w = \begin{bmatrix} B^{-1} a^{n+1} \\ -1 \end{bmatrix} \tag{4.1}
$$

spans the null space of A, $A(\bar{w} + \lambda \delta w) = b$. The critical value $\bar{\lambda} > 0$ is found by examining the inequalities

$$
C(\Delta)(\bar{w} + \lambda \delta w) \geqq d(\Delta) \quad \text{or} \quad C(\Delta)(\bar{w} - \lambda \delta w) \geqq d(\Delta); \tag{4.2}
$$

then we obtain

$$
\bar{\bar{w}} = \bar{w} + \bar{\lambda} \delta w \quad \text{or} \quad \bar{\bar{w}} = \bar{w} - \bar{\lambda} \delta w
$$

and the $(\bar{\bar{x}}, \bar{\bar{y}}, \bar{\bar{t}})$ corresponding to $\bar{\bar{w}}$ is just the exit of $H^{-1}(0)$ at $\check{z}$. When $\bar{\lambda} \to +\infty$, and $k = n$, x, where

$$
x_i = \delta w_i / \delta w_{n+1} = \delta u_i / \delta t \quad (\text{for all } 1 \leqq i \leqq n), \tag{4.3}
$$

is an approximate solution to the system (3.3).

The iterative steps of a major cycle of our algorithm are as follows.

Step 1. Set $q = (0, 0, \cdots, 0)$, $I_o(q) = \varnothing$, $I_e(q) = \{1, 2, \cdots, n\}$, $\dim \check{\sigma} = k = 0$, $A = (m^1 m^2 \cdots m^n g(0))$, $w = 0$, Δ: $(n+2, p)$, $(p, n+3)$ (for all $1 \leqq p \leqq n$).

Step 2. Compute δw by (4.1) and find $\bar{\lambda} > 0$ by examining (4.2). If $\bar{\lambda} < +\infty$, we obtain a critical edge (p, p') and let $\bar{w} = \bar{w} + \bar{\lambda}\delta w$ and then go to Step 3. Otherwise we obtain an approximate solution x by (4.3) and a major cycle is completed.

Step 3. Update information about $\check{z}$, including q, A, B^{-1}, Δ, $\bar{w}$, etc., and return to Step 2.

4.2. Updating large pieces. Now we discuss the updating technique in various critical cases. Because the updating of Δ is similar to that in [13], except that $(n+1, i)$ and $(i, n+2)$ are replaced by $(n+2, i)$, $(i, n+3)$ and there do not exist $(0, n+1)$, $(n+1, n+2)$ here, it is unnecessary to go into details.

We store $I_o(q)$ in a doubly linked list, so that if $p, p' \in I_o(q)$ and $(p, p') \in \Delta$, p is placed before p' in the list. We need not use the vectors t^p $(p = 1, 2, \cdots, n+1)$ which are used in [13] because new columns of A are easily generated by updating the order of $I_o(q)$. The technique for updating information in various critical cases is as follows.

Case 1. $I_e(q) = \varnothing$ and $\bar{\lambda} = +\infty$.

Then $x = (x_i) = (\delta w_i / \delta w_{n+1})$, where $i = 1, 2, \cdots, n$, is an approximate solution to the system $g(x) = 0$ and a major cycle is completed.

Case 2. The critical edge is $(n+2, p)$ for some $1 \leqq p \leqq n$. Then $\dim \check{\sigma} = k$, $\dim \tilde{\sigma} = k+1$ ($\tilde{\sigma}$ follows $\check{\sigma}$ in the iterative process).

Set $s_p = 1$, $k = k+1$. Because

$$x^{k-1} = x^0 + \varepsilon \sum_{i \in I_o(q)} s_i e^i, \qquad g(x^{k-1}) = g(x^0) + \varepsilon \sum_{i \in I_o(q)} s_i a^i.$$

Evaluate $g(x^k) = g(x^{k-1} + \varepsilon s_p e^p)$ and set $a^p = (g(x^k) - g(x^{k-1}))/\varepsilon s_p$ and $a^{n+1} = a^{n+1} - a^p x_p^0$; then update B^{-1}. Set $q_p = q_p + \varepsilon s_p$ and replace $\bar{w}_p = \bar{v}_p$ by $\bar{u}_p$, where $\bar{u}_p = x_p^0 \bar{w}_{n+1}$. Finally, we update Δ in a similar way to that in [13], set $I_o(q) = I_o(q) \cup \{p\}$, and make p be the last element in $I_o(q)$.

Case 3. The critical edge is $(p, n+3)$ for some $1 \leqq p \leqq n$. Proceed exactly as in Case 2, but with $s_p = -1$.

Case 4. The critical edge is $(p, n+1)$ for some $1 \leqq p \leqq n$. This is the reverse of Case 2 or 3. Then $\dim \check{\sigma} = k$, $\dim \tilde{\sigma} = k-1$.

Set $k = k-1$ and replace a^{n+1} by $a^{n+1} + a^p x_p^0$ and a^p by m^p; then update B^{-1}. Set $q_p = q_p - \varepsilon s_p$ and replace $\bar{w}_p = \bar{u}_p$ by $\bar{v}_p$, where $\bar{v}_p = q_p - \varepsilon s_p$ (the new q_p). Finally, we update Δ in a way similar to that in [13] and set $I_o(q) = I_o(q) \backslash \{p\}$, keeping the order of the remaining indices unchanged.

Case 5. The critical edge is $(0, p)$ for some $1 \leqq p \leqq n$. Then $\dim \check{\sigma} = \dim \tilde{\sigma} = k$. Set $g_{\text{old}} = g(x^0)$, $x^0 = x^0 + 2\varepsilon s_p e^p$ and evaluate $g(x^0)$. Set $s_p = -s_p$ and $a_{\text{old}}^p = a^p$, and replace $a^p = -(a_{\text{old}}^p + (g(x^0) - g_{\text{old}})/\varepsilon s_p)$, $a^{n+1} = a^{n+1} + (a_{\text{old}}^p - a^p)(x_p^0 + \varepsilon s_p)$; then update B^{-1}. The other information does not change.

Case 6. The critical edge is (p, p') for some $1 \leqq p, p' \leqq n$. Then $\dim \check{\sigma} = \dim \tilde{\sigma} = k$. First, we update Δ in a way identical to that in [13], and obtain the ordered lists $B_{p'}$, $\bar{B}_{p'}$, and $\bar{A}_{p'}$:

$$B_{p'} = \{i\colon i \text{ is in } I_o(q) \text{ and there is a path from } i \text{ to } p'\},$$

$$\bar{B}_{p'} = \{i\colon i \text{ is between } p \text{ and } p' \text{ in } I_o(q) \text{ and } i \in B_{p'}\},$$

$$\bar{A}_{p'} = \{i\colon i \text{ is between } p \text{ and } p' \text{ in } I_o(q) \text{ and } i \notin B_{p'}\}.$$

Next, we update $I_o(q)$ and A as follows:

Change the partial list $p \cdots p'$ of $I_o(q)$ to $\bar{B}_{p'} p' p \bar{A}_{p'}$.

Set

$$a_{\text{old}}^{p'} = a^{p'}, \qquad x' = x^0 + \varepsilon \sum_{i \in B_{p'}} s_i e^i, \qquad x'' = x' + s_{p'} e^{p'}, \qquad g(x') = g(x^0) + \varepsilon \sum_{i \in B_{p'}} s_i a^i.$$

Evaluate $g(x'')$ and set

$$a^{p'} = (g(x'') - g(x'))/\varepsilon s_{p'},$$
$$a^{p} = a^{p} + (a^{p'}_{\text{old}} - a^{p'}) s_{p'}/s_p,$$
$$a^{n+1} = a^{n+1} + (a^{p'}_{\text{old}} - a^{p'})(x^0_{p'} - (s_{p'}/s_p)x^0_p).$$

Finally, we update B^{-1}.

In various cases we can easily update B^{-1} as follows.

Cases 2-5. The p'th and the $n+1$'th columns of A have changed. Denote B of $\check{z}$ and $\tilde{B}$ of $\tilde{z}$ as

$$B = (a^1 a^2 \cdots a^p \cdots a^n),$$
$$\tilde{B} = (a^1 a^2 \cdots \tilde{a}^p \cdots a^n),$$

and

$$B^{-1} = (b^1 b^2 \cdots b^p \cdots b^n)^T. \tag{4.4}$$

Because $B^{-1}\tilde{B} = (e^1 e^2 \cdots B^{-1}\tilde{a}^p \cdots e^n)$, $\tilde{B}^{-1}$ can be obtained by pivoting in B^{-1}, using $(B^{-1}\tilde{a}^p)_p$ as the pivot element.

Case 6. The p'th, p''th, and $n+1$'th columns of A have changed and

$$\begin{aligned} B &= (a^1 a^2 \cdots a^p \cdots a^{p'} \cdots a^n), \\ \tilde{B} &= (a^1 a^2 \cdots a^p + (s_{p'}/s_p)a^{p'} - (s_{p'}/s_p)\tilde{a}^{p'} \cdots \tilde{a}^{p'} \cdots a^n). \end{aligned} \tag{4.5}$$

Denote

$$B^{-1} = (b^1 b^2 \cdots b^p \cdots b^{p'} \cdots b^n)^T. \tag{4.6}$$

If

$$B_1 = (a^1 a^2 \cdots a^p + (s_{p'}/s_p)a^{p'} \cdots a^{p'} \cdots a^n)$$

$$= B \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & +s_{p'}/s_p & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ & & p & & p' & & \end{bmatrix} \begin{matrix} \\ \\ p \\ \\ p' \\ \\ \end{matrix} \tag{4.7}$$

then

$$B_1^{-1} = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & -s_{p'}/s_p & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix} B \tag{4.8}$$

$$= (b^1 b^2 \cdots b^p \cdots b^{p'} - (s_{p'}/s_p)b^p \cdots b^n)^T.$$

Now the inverse

$$B_2^{-1} = (\tilde{b}^1 \tilde{b}^2 \cdots \tilde{b}^p \cdots \tilde{b}^{p'} \cdots \tilde{b}^n)^T \tag{4.9}$$

of B_2, where

$$B_2 = (a^1 a^2 \cdots a^p + (s_{p'}/s_p) a^{p'} \cdots \tilde{a}^{p'} \cdots a^n)$$

can be obtained by using the same method as in Cases 2–5. Then, as in (4.7) and (4.8), we find that

$$\tilde{B}^{-1} = (\tilde{b}^1 \tilde{b}^2 \cdots \tilde{b}^p \cdots \tilde{b}^{p'} + (s_{p'}/s_p) \tilde{b}^p \cdots \tilde{b}^n)^T.$$

We now have Theorem 3.

THEOREM 3. *In Case* 6, *updating* B^{-1} *can be performed as follows*:
Step 1. *Obtain* B_1^{-1} *by replacing the* p*"th row of* B^{-1} *by* $b^{p'} - (s_{p'}/s_p) b^p$.
Step 2. *Obtain* B_2^{-1} *by pivoting in* B_1^{-1} *using* $(B_1^{-1} \tilde{a}^{p'})_{p'}$ *as the pivot element.*
Step 3. *Obtain the new* B^{-1} *by replacing the* p*"th row of* B_2^{-1} *by* $\tilde{b}^{p'} + (s_{p'}/s_p) \tilde{b}^p$.
Here $\tilde{a}^{p'}$, b^p, $b^{p'}$, B_1, B_2, $\tilde{b}^p$, $\tilde{b}^{p'}$ are defined by (4.4)–(4.7) and (4.9).

4.3. Some revisions. In § 3 of [13], in order to update large pieces, $I = \{i: \pi^{-1}(i) < j+1\}$ is stored in a doubly linked list, so that if π is the permutation corresponding to the ordered list I, then $\hat{j}_1((y, 1), \pi, s)$ is contained in the current piece $\check{\sigma}$. For generating new columns of A, the vector t^p of 0's and ± 1's is used. If $1 \leqq \pi^{-1}(p) < j$, then $a^p = (f(y + \varepsilon t^p + \varepsilon s_p e^p) - f(y + \varepsilon t^p))/\varepsilon s_p$ and

$$a^{n+1} = f(y + \varepsilon t^{n+1}) - f^0(y + \varepsilon t^{n+1}).$$

From our computational experience, some details about updating t^p and A are suspected to be mistaken.

First, we found that the updating of the order of I in Case 6 was lost. We cannot correctly update the graph Δ without the correct list I.

Next, suppose that the order of I is correct. Because of the special structure of f, in Case 4, there may be some nodes k between p and $n+1$ in I and all these t^k's, not only t^{n+1}, need to be updated. In Case 6, the updating of t^p's is more complex. If the critical edge is (p, q), there may be three kinds of nodes between p and q:

$$\bar{A}_p = \{i: i \text{ is between } p \text{ and } q \text{ in } I \text{ and there is a path from } p \text{ to } i\},$$

$$\bar{B}_q = \{i: i \text{ is between } p \text{ and } q \text{ in } I \text{ and there is a path from } i \text{ to } q\},$$

$$S = \{i: i \text{ is between } p \text{ and } q \text{ in } I,\ i \notin \bar{A}_p,\ i \notin \bar{B}_q\}.$$

In the list I of the adjacent large piece $\tilde{\sigma}$, $\bar{B}_q$ must be kept before q. Thus we should not only update t^q and t^p but also update t^k for all $k \in \bar{B}_q$. Clearly, the computations of t^q and corresponding a^q, as reported in [13], are mistaken. In some of the iterative steps, for example, the subgraph not including nodes $n+2$ and $n+3$ of Δ might be as in Fig. 4.1. Then $\check{\sigma}$ is a complex consisting of two simplices which correspond to the subgraphs shown in Fig. 4.2, and the t^p must be corresponding to the π of one of the

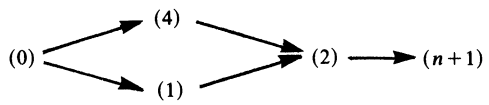


FIG. 4.1

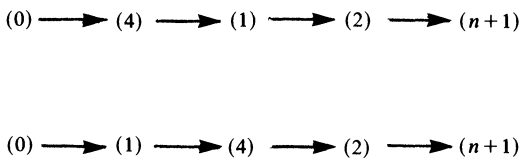


FIG. 4.2

two. If the critical edge is (4, 2), the subgraph of the adjacent large piece $\tilde{\sigma}$ is as in Fig. 4.3. Suppose that $s_1 = 1$, $s_2 = -1$, $s_4 = -1$, and that t^1, t^2, and t^4 are

$$t^1 = (0, 0, 0, -1, 0, \cdots, 0)^T,$$
$$t^2 = (1, 0, 0, -1, 0, \cdots, 0)^T,$$
$$t^4 = (0, 0, 0, 0, 0, \cdots, 0)^T;$$

then they should become

$$t^1 = (0, 0, 0, 0, 0, \cdots, 0)^T,$$
$$t^2 = (1, 0, 0, 0, 0, \cdots, 0)^T,$$
$$t^4 = (1, -1, 0, 0, 0, \cdots, 0)^T,$$

not $t^2 \leftarrow t^4$ and $t^4 \leftarrow t^4 + s_2 e^2$, according to $t^q \leftarrow t^p$ and $t^p \leftarrow t^p + s_q e^q$, as reported in [13].

It seems to us that the mistakes can be corrected as follows. Do not store the t^p's, but correctly update the list I. Cases 1–5 are the same as in [13], except that a^p has the wrong sign in Case 5. In Case 6, remove ordered list $[\bar{B}_q \quad q]$ to before p. That is, the order I may be of the form:

$$I: \cdots p \bar{B}_q S \bar{A}_p q \cdots,$$

where the elements in $\bar{B}_q$, S, $\bar{A}_p$ may be reordered to some extent. We change I to

$$\tilde{I}: \cdots \bar{B}_q q p S \bar{A}_p \cdots$$

and change Δ accordingly. Let J be the set of indices in $\tilde{I}$ up to but not including q. Then we know that

$$f\left(y + \varepsilon \sum_{j \in J} s_j e^j\right) = f(y) + \varepsilon \sum_{j \in J} s_j a^j,$$

and we evaluate $f(y + \varepsilon \sum_{j \in J} s_j e^j + s_q e^q)$. Set

$$a^q_{\text{old}} = a^q, \qquad a^q = \left(f\left(y + \varepsilon \sum_{j \in J} s_j e^j + s_q e^q\right) - f\left(y + \varepsilon \sum_{j \in J} s_j e^j\right)\right) \Big/ \varepsilon s_q,$$
$$a^p = a^p + (a^q_{\text{old}} - a^q) s_q / s_p.$$

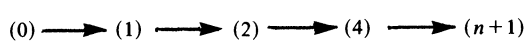


FIG. 4.3

5. Computational results. In order to prove how efficient our method is, we have made a program of the method in FORTRAN 77. For the examples we used, Problems 1-7 and 11 are chosen from those in the MINPACK collection [7], but the forms of Problems 1 and 3 are slightly changed. Problem 8 is from [5], and Problems 9-10 are original.

PROBLEM 1. Rosenbrock function:

$$f_1(x) = 10(x_1 - x_2^2),$$
$$f_2(x) = x_2 - 1.$$

PROBLEM 2. Freudenstein and Roth function:

$$f_1(x) = -13 + x_1 + ((5 - x_2)x_2 - 2)x_2,$$
$$f_2(x) = -29 + x_1 + ((x_2 + 1)x_2 - 14)x_2.$$

PROBLEM 3. Extended Rosenbrock function:

$$f_{2i-1}(x) = 10(x_{2i-1} - x_{2i}^2),$$
$$f_{2i}(x) = x_{2i} - 1.$$

PROBLEM 4. Discrete boundary value function:

$$f_i(x) = 2x_i - x_{i-1} - x_{i+1} + h^2(x_i + t_i + 1)^3/2$$

where $h = 1/(n+1)$, $t_i = ih$, and $x_0 = x_{n+1} = 0$.

PROBLEM 5. Discrete integral equation function:

$$f_i(x) = x_i + h\left[(1 - t_i)\sum_{j=1}^{i} t_j(x_j + t_j + 1)^3 + t_i \sum_{j=i+1}^{n} (1 - t_j)(x_j + t_j + 1)^3\right]\Big/2$$

where $h = 1/(n+1)$, $t_i = ih$, and $x_0 = x_{n+1} = 0$.

PROBLEM 6. Broyden tridiagonal function:

$$f_i(x) = (3 - 2x_i)x_i - x_{i-1} - 2x_{i+1} + 1$$

where $x_0 = x_{n+1} = 0$.

PROBLEM 7. Broyden banded function:

$$f_i(x) = x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j)$$

where $J_i = \{j: j \neq i, \max(1, i - m_1) \leqq j \leqq \min(n, i + m_u)\}$ and $m_1 = 5$, $m_u = 1$.

PROBLEM 8. Separable cubic function:

$$f_i(x) = x_i - \left(i + \sum_{j=1}^{n} x_j^3\right)\Big/2n.$$

PROBLEM 9. Nearly separable function:

$$f_i(x) = x_i + \sum_{j=1}^{n} x_j^3 + \cos(x_{i-1} + x_{i+1})$$

where $x_0 = x_{n+1} = 0$.

PROBLEM 10. Tridiagonal function:

$$f_i(x) = x_i - \exp(\cos(i(x_{i-1} + x_i + x_{i+1})))$$

where $x_0 = x_{n+1} = 0$.

PROBLEM 11. Trigonometric function:

$$f_i(x) = n - \sum_{j=1}^{n} \cos x_j + i(1 - \cos x_i) - \sin x_i.$$

In all runs, we used $M = I$ throughout and fixed $\varepsilon_{k+1} = \varepsilon_k/2$. We stopped the computations when we obtained an approximate solution, x, satisfying $\|f(x)\| \leqq 10^{-8}$ (except that it was 10^{-4} in Problem 11). The computational results for all the problems obtained in two ways—considering and not considering the structure of f—are shown in Tables 5.1-5.7. In these tables, the numbers to the left of "/" are the numbers of pivoting operations and the numbers to the right of "/" are the numbers of evaluations of the functions. From these results we can see that the algorithm considering the structure of f greatly saves the number of pivoting operations and the number of evaluations of the functions. But we also know that the complexity of our large-piece implementation depends on the number of facets of such a piece, $\check{\sigma}$, or, equivalently, that it depends on the number of edges of a graph Δ. In [13] it is proved that the number can be no more than $O(n^{3/2})$. For our examples, because Problems 1-8 and 11 are separable, the numbers are $2n$, and the total numbers of facets for Problems 9 and 10 are listed in Tables 5.5 and 5.6 as "total ratios" columns. We can see that their averages per pivot are generally no more than $2n$. In the case not considering structure, $\hat{\sigma}$ has $2n - k + 1$ faces. From these results we can conclude that the algorithm exploiting the structure of f often greatly saves the computations, in particular, the computations

TABLE 5.1
Computational results for Problems 1 *and* 2.

Problem	w^0	ε_1	Not considering structure	Considering structure
1	(1.0, −1.2)	0.5	22/20	17/15
		1.0	15/14	12/11
		2.0	11/11	10/10
	(10.0, −12.0)	5.0	99/78	77/56
		10.0	61/49	49/37
		20.0	33/27	30/24
	(100.0, −120.0)	50.0	966/724	735/493
		100.0	590/445	451/306
		200.0	297/209	277/189
2	(0.5, −2.0)	0.5	63/50	47/34
		1.0	37/30	28/21
		2.0	20/17	16/13
	(5.0, −20.0)	5.0	87/87	65/65
		10.0	64/71	49/56
		20.0	65/74	49/58
	(50.0, −200.0)	50.0	78/85	60/67
		100.0	71/82	54/65
		200.0	74/87	55/68

TABLE 5.2
Computational results for Problem 3 ($w^0_{2i-1} = 1$, $w^0_{2i} = -1.2$).

	$\varepsilon_1 = 0.5$		$\varepsilon_1 = 2.0$	
n	Not considering structure	Considering structure	Not considering structure	Considering structure
2	22/20	17/15	11/11	10/10
4	59/53	32/26	24/22	18/16
6	115/107	47/37	42/38	26/22
8	189/175	62/48	65/59	34/28
10	286/283	77/59	93/85	42/34
12	387/365	92/70	126/116	50/40
14	519/493	107/81	164/152	58/46
16	665/635	122/92	207/193	66/52
18	823/789	137/103	255/239	74/58
20	1002/964	152/114	308/290	82/64

TABLE 5.3
Computational results for Problems 4–7 ($\varepsilon_1 = 0.5$).

n	Not considering structure	Considering structure	Not considering structure	Considering structure
	Problem 4 ($w^0_j = t_j(t_j - 1)$)		Problem 5 ($w^0_j = t_j(t_j - 1)$)	
5	34/39	30/35	33/38	30/35
10	89/93	57/61	61/66	55/60
15	130/132	68/70	93/98	80/85
20	215/216	90/91	126/131	105/110
25	327/327	112/112	160/165	130/135
30	433/432	134/133	200/205	155/160
	Problem 6 ($w^0_j = -1$)		Problem 7 ($w^0_j = -1$)	
5	55/62	43/50	67/75	50/58
10	94/101	78/85	171/179	90/98
15	145/152	113/120	384/392	130/138
20	185/192	148/155	764/772	170/178
25	234/241	183/190	1317/1325	210/218
30	270/277	218/225	2022/2030	250/258

TABLE 5.4
Computational results for Problem 8 ($\varepsilon_1 = 0.5$, $w^0_j = -0.5(n+1-i)/(n+1)$).

	Total operation		First major cycle	
n	Not considering structure	Considering structure	Not considering structure	Considering structure
5	51/57	41/47	21/22	11/12
10	121/127	76/82	66/67	21/22
15	216/222	111/117	136/137	31/32
20	336/342	146/152	231/232	41/42
25	481/487	181/187	351/352	51/52
30	651/657	216/222	496/497	61/62

TABLE 5.5
Computational results for Problem 9 ($\varepsilon_1 = 0.5$, $w_j^0 = 0.5(n+1-i)/(n+1)$).

n	Not considering structure	Considering structure	Total ratios
5	87/95	53/61	567
10	413/407	185/179	3,832
15	562/562	193/193	5,412
20	1536/1504	387/355	14,859
25	1436/1436	362/362	15,722
30	3121/3077	581/537	30,623

TABLE 5.6
Computational results for Problem 10 ($\varepsilon_1 = 0.5$, $w_j^0 = -0.5(n+1-i)/(n+1)$).

	Total operation		First major cycle		
n	Not considering structure	Considering structure	Not considering structure	Considering structure	Total ratios
3	81/82	81/82	38/31	38/31	442
4	155/153	140/138	78/70	71/63	989
5	248/243	197/192	127/115	99/87	1,926
6	711/642	549/480	360/304	280/224	6,789
7	1338/1218	893/773	894/796	585/487	11,839
8	3384/3057	2048/1714	2439/2184	1393/1138	31,633
9	4673/4285	2484/2096	3450/3142	1739/1431	42,885
10	4174/3899	2112/1837	2675/2464	1238/1027	40,282

TABLE 5.7
Computational results for Problem 11 ($w_j^0 = 1/n$).

ε_1	n	Not considering structure	Considering structure	The solutions
0.25	1	6/9	6/9	(.927212D+00)
	2	501/387	386/372	(−.628318D+01, −.125664D+02)
	3	55902/44488	14808/10088	(−.501268D+02, −.752458D+02, −.100063D+03)
0.5	1	8/12	8/12	(.927243D+00)
	2	660/509	507/356	(−.125664D+02, −.251327D+02)
	3	18252/14557	11629/7934	(−.501268D+02, −.752458D+02, −.106346D+03)

of the first major cycle (see Tables 5.4 and 5.6). The higher the dimension, the more remarkable the improvement. Obviously, if we use a technique saving the number of major cycles, such as Saigal's acceleration technique [8] or joining a quasi-Newton method ([14]), the computational efficiency will be more greatly improved.

Acknowledgments. We are most grateful to Professor M. Kojima, Professor M. J. Todd, and the referees for several very helpful comments concerning this work. Particularly, the corrections shown at the end of § 4.3 were suggested by Professor Todd. We are also grateful to Liu Chao-yang and Professor Li He for their kind help with this paper.

REFERENCES

[1] E. ALLGOWER AND K. GEORG, *Simplicial and continuation methods for approximating fixed points and solutions to systems of equations*, SIAM Rev., 22 (1980), pp. 28–85.

[2] R. FREUND, *Variable-dimension complexes with applications*, Math. Oper. Res., 9 (1984), pp. 479–497.

[3] M. KOJIMA, *On the homotopic approach to systems of equations with separable mappings*, Math. Programming Stud., 7 (1978), pp. 170–184.

[4] M. KOJIMA AND Y. YAMAMOTO, *Variable dimension algorithms: Basic theory, interpretations and extensions of some existing methods*, Math. Programming, 24 (1982), pp. 177–215.

[5] ———, *A unified approach to several restart fixed point algorithms for their implementation and a new variable dimension algorithm*, Math. Programming, 28 (1984), pp. 288–328.

[6] G. VAN DER LAAN AND A. J. J. TALMAN, *A restart algorithm for computing fixed point without extra dimension*, Math. Programming, 17 (1979), pp. 74–84.

[7] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.

[8] R. SAIGAL, *On the convergence rate of algorithms for solving equations that are based on methods of complementarity pivoting*, Math. Oper. Res., 2 (1977), pp. 108–124.

[9] M. J. TODD, *Fixed-point algorithms that allow restarting without an extra dimension*, Tech. Report No. 379, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, September 1978.

[10] ———, *Traversing large pieces of linearity in algorithms that solve equations by following piecewise-linear paths*, Math. Oper. Res., 5 (1980), pp. 242–257.

[11] ———, *Exploiting structure in piecewise-linear homotopy algorithms for solving equations*, Math. Programming, 18 (1980), pp. 233–247.

[12] ———, *Numerical stability and sparsity in piecewise-linear algorithms*, in Analysis and Computation of Fixed Points, S. M. Robinson, ed., Academic Press, New York, 1980, pp. 1–24.

[13] ———, *Piecewise-linear homotopy algorithms for sparse systems of nonlinear equations*, SIAM J. Control Optim., 21 (1983), pp. 204–214.

[14] B. YANG AND M. KOJIMA, *Improving the computational efficiency of fixed point algorithms*, J. Oper. Res. Soc. Japan, 271 (1984), pp. 59–76.

[15] H. SCARF, *The approximation of fixed points of a continuous mapping*, SIAM J. Appl. Math., 15 (1967), pp. 1328–1343.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 222–251, May 1991

005

EXPERIMENTS WITH CONJUGATE GRADIENT ALGORITHMS FOR HOMOTOPY CURVE TRACKING*

KASHMIRA M. IRANI†, MANOHAR P. KAMAT‡, CALVIN J. RIBBENS†, HOMER F. WALKER§, AND LAYNE T. WATSON†

Abstract. There are algorithms for finding zeros or fixed points of nonlinear systems of equations that are globally convergent for almost all starting points, i.e., with probability one. The essence of all such algorithms is the construction of an appropriate homotopy map and then tracking some smooth curve in the zero set of this homotopy map. HOMPACK is a mathematical software package implementing globally convergent homotopy algorithms with three different techniques for tracking a homotopy zero curve, and has separate routines for dense and sparse Jacobian matrices. The HOMPACK algorithms for sparse Jacobian matrices use a preconditioned conjugate gradient algorithm for the computation of the kernel of the homotopy Jacobian matrix, a required linear algebra step for homotopy curve tracking. Here variants of the conjugate gradient algorithm are implemented in the context of homotopy curve tracking and compared with Craig's preconditioned conjugate gradient method used in HOMPACK. The test problems used include actual large scale, sparse structural mechanics problems.

Key words. globally convergent, homotopy algorithm, nonlinear equations, preconditioned conjugate gradient, homotopy curve tracking, sparse matrix, matrix splitting, bordered matrix

AMS(MOS) subject classifications. 65F10, 65F50, 65H10, 65K10

1. Introduction.

The fundamental problem motivating this work is to solve a nonlinear system of equations $F(x) = 0$, where $F : E^n \to E^n$ is a C^2 map defined on real n-dimensional Euclidean space E^n. The homotopy approach to solving $F(x) = 0$ is to construct a continuous map $H(\lambda, x)$, the "homotopy," deforming a simple function $s(x)$ to the given function $F(x)$ as λ varies from 0 to 1. Starting from the easily obtained solution to $H(0, x) = s(x) = 0$, the essence of a homotopy algorithm is to track solutions of $H(\lambda, x) = 0$ until a solution of $H(1, x) = F(x) = 0$ is obtained. The theoretical and implementational details of such algorithms are nontrivial, and significant progress on both aspects has been made recently [37], [52].

* Received by the editors December 12, 1989; accepted for publication (in revised form) October 15, 1990.

† Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061. The work of these authors was supported in part by Department of Energy grant DE-FG05-88ER25068, National Aeronautics and Space Administration grant NAG-1-1079, National Aeronautics and Space Administration contract NAS1-18471-24, National Science Foundation grant CTS-8913198, and Air Force Office of Scientific Research grant 89–0497.

‡ School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, Georgia 30332.

§ Department of Mathematics and Statistics, Utah State University, Logan, Utah 84322. The work of this author was supported in part by Department of Energy grant DE-FG02-86ER25018, Army grant DAAL03-88-K, and National Science Foundation grant DMS-8800995.

Homotopies are a traditional part of topology, and only recently have begun to be used for practical numerical computation. The (globally convergent probability-one) homotopies considered here are sometimes called "artificial-parameter generic homotopies," in contrast to natural-parameter homotopies, where the homotopy variable is a physically meaningful parameter. In the latter case, which is frequently of interest, the resulting homotopy zero curves must be dealt with as they are, bifurcations, ill-conditioning, etc. The homotopy zero curves for artificial-parameter generic homotopies obey strict smoothness conditions, which generally will not hold if the homotopy parameter represents a physically meaningful quantity, but they can always be obtained via certain generic constructions using an artificial (i.e., nonphysical) homotopy parameter. Not just any random perturbation will suffice to create a globally convergent probability-one (generic) homotopy, e.g., the perturbation implied by discretization is generally not sufficient to produce a probability-one homotopy map.

If the objective is to solve a "parameter-free" system of equations, $F(x) = 0$, then extra attention can be devoted to constructing the homotopy, and the curve-tracking algorithm can be limited to a well-behaved class of curves. The goal of using these globally convergent probability-one homotopies is to solve fixed-point and zero-finding problems with homotopies whose zero curves do not have bifurcations and other singular and ill-conditioned behavior. The mathematical software package HOMPACK, used here for comparative purposes, is designed for globally convergent probability-one homotopies.

The theory and algorithms for functions $F(x)$ with small dense Jacobian matrices $DF(x)$ are well developed, which is not the case for large sparse $DF(x)$, the topic of this paper. Solving large sparse nonlinear systems of equations via homotopy methods involves sparse rectangular linear systems of equations and iterative methods for the solution of such sparse systems. Preconditioning techniques are used to make the iterative methods more efficient.

Section 2 discusses the zero-finding problem and the normal flow homotopy algorithm. Section 3 introduces iterative methods for solving invertible linear systems. Section 4 discusses the linear algebra details of homotopy curve tracking and various algorithmic possibilities for that. Section 5 presents the numerical results of the implementation of the various algorithms on several test problems. Some general conclusions from these results are drawn in §6.

2. Globally convergent homotopy algorithms. The philosophy of globally convergent probability-one homotopy algorithms is to create homotopies whose zero curves are well behaved with well-conditioned Jacobian matrices and that reach a solution for almost all choices of a parameter. These homotopies are used to solve fixed-point and zero-finding problems.

Let B be the closed unit ball in n-dimensional real Euclidean space E^n, and let $f : B \to B$ be a C^2 map. The fixed-point problem is to solve $x = f(x)$. Define $\rho_a : [0,1) \times B \to E^n$ by

$$\rho_a(\lambda, x) = \lambda(x - f(x)) + (1 - \lambda)(x - a). \tag{1}$$

The fundamental result [10] is that for almost all a in the interior of B, there is a zero curve $\gamma \subset [0,1) \times B$ of ρ_a, along which the Jacobian matrix $D\rho_a(\lambda, x)$ has rank n,

emanating from $(0, a)$, and reaching a point $(1, \bar{x})$, where $\bar{x}$ is a fixed point of f. Thus with probability one, picking a starting point $a \in \text{int } B$ and following γ leads to a fixed point $\bar{x}$ of f. An important distinction between standard continuation and modern probability-one homotopy algorithms is that for the latter λ is not necessarily monotonically increasing along γ. Indeed, part of the power of probability-one homotopy algorithms derives from the lack of a monotonicity requirement for λ.

The zero-finding problem

$$F(x) = 0, \tag{2}$$

where $F : E^n \to E^n$ is a C^2 map, is more complicated. Suppose that there exists a C^2 map

$$\rho : E^m \times [0, 1) \times E^n \to E^n$$

such that

(a) the $n \times (m + 1 + n)$ Jacobian matrix $D\rho(a, \lambda, x)$ has rank n on the set

$$\rho^{-1}(0) = \{(a, \lambda, x) \mid |a \in E^m, 0 \leq \lambda < 1, x \in E^n, \rho(a, \lambda, x) = 0\},$$

and for any fixed $a \in E^m$, letting $\rho_a(\lambda, x) = \rho(a, \lambda, x)$,

(b) $\rho_a(0, x) = \rho(a, 0, x) = 0$ has a unique solution x_0,

(c) $\rho_a(1, x) = F(x)$,

(d) $\rho_a^{-1}(0)$ is bounded.

Then for almost all $a \in E^m$ there exists a zero curve γ of ρ_a along which the Jacobian matrix $D\rho_a$ has rank n, emanating from $(0, x_0)$ and reaching a zero $\bar{x}$ of F at $\lambda = 1$. γ does not intersect itself and is disjoint from any other zeros of ρ_a. The globally convergent homotopy algorithm is to pick $a \in E^m$ (which uniquely determines x_0), and then track the homotopy zero curve γ starting at $(0, x_0)$ until the point $(1, \bar{x})$ is reached.

There are many different algorithms for tracking the zero curve γ; the mathematical software package HOMPACK [52], [53] supports three such algorithms: ordinary differential equation-based, normal flow, and augmented Jacobian matrix. Small dense and large sparse Jacobian matrices require substantially different algorithms. Large nonlinear systems of equations with sparse symmetric Jacobian matrices occur in many engineering disciplines (the symmetry in the problems of interest here is due to the fact that the Jacobian matrix is actually the Hessian of a potential energy function). In this paper, we consider only the zero finding problem $F(x) = 0$, the normal flow curve tracking algorithm, and large sparse symmetric Jacobian matrices $DF(x)$ stored in a packed skyline data structure.

Consider the homotopy map

$$\rho_a(x, \lambda) = \lambda F(x) + (1 - \lambda)(x - a). \tag{3}$$

The matrix $D_x\rho_a(x, \lambda) = \lambda DF(x) + (1 - \lambda)I$ is symmetric and sparse with a "skyline" structure. Such matrices are typically stored in packed skyline format, in which the upper triangle is stored in a one-dimensional indexed array. An auxiliary array of diagonal indices is also required. Assuming that $F(x)$ is C^2, a is such that the Jacobian

matrix $D\rho_a(x,\lambda)$ has full rank along γ, and γ is bounded, the zero curve γ is C^1 and can be parameterized by arc length s. Thus $x = x(s), \lambda = \lambda(s)$ along γ, and

$$\rho_a(x(s),\lambda(s)) = 0$$

identically in s.

The zero curve γ given by $(x(s),\lambda(s))$ is the trajectory of the initial value problem

$$\frac{d}{ds}\rho_a(x(s),\lambda(s)) = [D_x\rho_a(x(s),\lambda(s)), D_\lambda\rho_a(x(s),\lambda(s))]\begin{pmatrix} dx/ds \\ d\lambda/ds \end{pmatrix} = 0, \tag{4}$$

$$\left\|\left(\frac{dx}{ds},\frac{d\lambda}{ds}\right)\right\|_2 = 1, \tag{5}$$

$$x(0) = a, \quad \lambda(0) = 0. \tag{6}$$

Since the Jacobian matrix has rank n along γ, the derivative $(dx/ds, d\lambda/ds)$ is uniquely determined by (4), (5) and continuity, and the initial value problem (4)–(6) can be solved for $x(s),\lambda(s)$. From (4) it can be seen that the unit tangent $(dx/ds, d\lambda/ds)$ to γ is in the kernel of $D\rho_a$.

The normal flow curve tracking algorithm has four phases: prediction, correction, step size estimation, and computation of the solution at $\lambda = 1$. For the prediction phase, assume that two points $P^{(1)} = (x(s_1),\lambda(s_1))$, $P^{(2)} = (x(s_2),\lambda(s_2))$ on γ with corresponding tangent vectors $(dx/ds(s_1),\ d\lambda/ds(s_1))$, $(dx/ds(s_2),\ d\lambda/ds(s_2))$ have been found, and h is an estimate of the optimal step (in arc length) to take along γ. The prediction of the next point on γ is

$$Z^{(0)} = p(s_2 + h), \tag{7}$$

where $p(s)$ is the Hermite cubic interpolating $(x(s),\lambda(s))$ at s_1 and s_2. Precisely,

$$\begin{aligned} p(s_1) &= (x(s_1),\lambda(s_1)), \quad & p'(s_1) &= (dx/ds(s_1), d\lambda/ds(s_1)), \\ p(s_2) &= (x(s_2),\lambda(s_2)), \quad & p'(s_2) &= (dx/ds(s_2), d\lambda/ds(s_2)), \end{aligned}$$

and each component of $p(s)$ is a polynomial in s of degree less than or equal to 3.

Starting at the predicted point $Z^{(0)}$, the corrector iteration is

$$Z^{(k+1)} = Z^{(k)} - \left[D\rho_a\left(Z^{(k)}\right)\right]^+ \rho_a\left(Z^{(k)}\right), \qquad k = 0,1,\cdots, \tag{8}$$

where $\left[D\rho_a(Z^{(k)})\right]^+$ is the Moore–Penrose pseudoinverse of the $n \times (n+1)$ Jacobian matrix $D\rho_a$. Small perturbations of a produce small changes in the trajectory γ, and the family of trajectories γ for varying a is known as the "Davidenko flow." Geometrically, the iterates given by (8) return to the zero curve along the flow normal to the Davidenko flow, hence the name "normal flow algorithm."

A corrector step ΔZ is the unique minimum norm solution of the equation

$$[D\rho_a]\Delta Z = -\rho_a. \tag{9}$$

Fortunately ΔZ can be calculated at the same time as the kernel of $[D\rho_a]$, and with just a little more work. The numerical linear algebra details for solving (9), the optimal step size estimation, and the endgame to obtain the solution at $\lambda = 1$ are in [52], [53].

The calculation of the implicitly defined derivative $(dx/ds, d\lambda/ds)$ is done by computing the one-dimensional kernel of $D\rho_a$, i.e., by solving the $n \times (n+1)$ linear system $[D\rho_a]y = 0$. This can be elegantly and efficiently done for small dense matrices [47], [51], but the large sparse Jacobian matrix presents special difficulties. The difficulty now is that the first n columns of the Jacobian matrix $D\rho_a(x, \lambda)$ involving $DF(x)$ are definitely special, and any attempt to treat all $n + 1$ columns uniformly would be disastrous from the point of view of storage allocation. Hence, what is required is a good algorithm for solving nonsquare linear systems of equations (9) where the leading $n \times n$ submatrix $D_x\rho_a$ of $D\rho_a$ is symmetric and sparse. This paper considers various iterative methods for solving such linear systems of equations.

3. Iterative methods for invertible linear systems. Nonsquare systems of the form (9), involved in the tangent vector and normal flow iteration calculations, are converted to equivalent square linear systems of the form

$$Ay = \begin{pmatrix} B & f \\ c^t & d \end{pmatrix} y = b, \tag{10}$$

where the $n \times n$ matrix B is bordered by the vectors f and c to form a larger system of dimension $(n + 1) \times (n + 1)$. In the present context $B = D_x\rho_a(x, \lambda)$ is symmetric and sparse, but A is not necessarily symmetric.

Iterative methods are used for solving these linear systems. (If B has only a couple nonpositive eigenvalues, direct methods are a viable alternative; this issue is addressed later.) Iterative methods compute a sequence of approximate solutions $\{x_i\}$ which converge to the exact solution x by some algorithm of the form

$$x_{i+1} = F_i(x_0, x_1, \cdots, x_i),$$

where x_0 is an arbitrary initial guess and F_i may be linear or nonlinear.

Iterative methods require the coefficient matrix A in the algorithm, generally only to compute matrix-vector products. Since matrix-vector computations are quite inexpensive for sparse problems, iterative methods have low computational cost per iteration. Iterative methods are also attractive because they have low storage requirements, due to the fact that at each iteration, only a small number of vectors of length $N = n+1$ need to be computed and stored to calculate the next iterate x_{i+1}, and A itself can be generated or stored compactly. Thus iterative methods are sometimes more attractive than direct methods for solving large sparse linear systems of equations.

Iterative methods such as the successive over-relaxation (SOR) algorithm [43] and the alternating direction implicit (ADI) algorithm [57] require the estimation of scalar parameters. The conjugate gradient procedure [24] is an efficient algorithm

for solving symmetric positive definite systems which requires no such estimates. For many years, the only iterative methods known to converge for general nonsymmetric problems were the conjugate gradient method applied to the normal equations [24] and Lanczos' biconjugate gradient algorithm [29]. Other early conjugate gradient-like methods for nonsymmetric problems which avoided the use of the normal equations were the generalized conjugate gradient method of Concus and Golub [11]–[12] and Widlund [56], and Orthomin by Vinsome [44]. These methods apply only to matrices with positive definite symmetric part, although with preconditioning they can in principle be used to solve more general problems [18]. Other conjugate gradient-like methods for more general problems were proposed by Axelsson [1]–[3]; Eisenstat, Elman, and Schultz [17]; Jea [25]; Saad [39]; Young and Jea [58]–[59]; and Saad and Schultz [41]. Preconditioning techniques that have been effective for symmetric, positive definite systems include the incomplete LU factorization [30], [31], the modified incomplete LU factorization [15], [22], and the SSOR preconditioning [57]. Most of these extend naturally to nonsymmetric problems. A lot of work has also been done comparing these various iterative methods and the preconditioning techniques [9], [14], [18], [41]. Unfortunately very little of this existing theory is directly applicable to the sparse linear systems arising from homotopy curve tracking, because they are nonsquare, generally indefinite, and lack special structure typical of PDE problems.

The rate of convergence of conjugate gradient-type methods depends on the symmetry, inertia, spectrum, and condition number of the coefficient matrix. There are efficient conjugate gradient algorithms for solving linear systems with symmetric positive definite coefficient matrices, whereas no comparable theory exists for general systems with nonsymmetric or indefinite A. This paper compares the relative performance of conjugate gradient-type algorithms for solving nonsymmetric or indefinite linear systems of the form $Ax = b$ arising from globally convergent homotopy algorithms, in terms of execution time, storage requirements, and the number of iterations required to converge.

Let Q be an $N \times N$ nonsingular matrix. The solution to $Ax = b$ can also be obtained by solving the system

$$\tilde{A}x = (Q^{-1}A)x = Q^{-1}b = \tilde{b}.$$

The use of such an auxiliary matrix is known as *preconditioning*. The goal of preconditioning is to decrease the computational effort required to solve linear systems of equations by increasing the rate of convergence of an iterative method. For preconditioning to be effective, the faster convergence must outweigh the costs of applying the preconditioning, so that the total cost of solving the linear system is lower. The preconditioned coefficient matrix $\tilde{A}$ is usually not explicitly computed or stored. The main reason for this is that although A is sparse, $\tilde{A}$ may not be. The extra work of preconditioning, then, occurs in the preconditioned matrix-vector products involving Q^{-1}. The main storage cost for preconditioning is usually for Q, which typically is stored, so that one extra array is required to handle the preconditioning operation.

As mentioned above, one iterative method known to converge for general nonsymmetric problems is the conjugate gradient method applied to the normal equations.

Given any nonsingular matrix A, the system of linear equations $Ay = b$ can be solved by considering the linear system (normal equations)

$$A^t Ay = A^t b,$$

or the similar system

$$AA^t z = b, \qquad y = A^t z.$$

Since the coefficient matrix for the latter system is both symmetric and positive definite, the system can be solved by the conjugate gradient algorithm. Once a solution vector z is obtained, the vector y from the original system can be computed as $y = A^t z$. The drawback of this technique is that, while the coefficient matrix is symmetric and positive definite, the convergence rate depends on $\text{cond}(AA^t) = (\text{cond}(A))^2$ rather than on $\text{cond}(A)$; see [18] for a precise statement.

An implementation of the conjugate gradient algorithm in which y is computed directly, without reference to z, any approximations of z, or AA^t is due to Craig [13] and is described in [19] and [23]. (Of course, the convergence rate still depends on $\text{cond}(AA^T) = (\text{cond}(A))^2$ in general.) Craig's preconditioned algorithm is:

choose y_0, Q;
set $r_0 = b - Ay_0$;
set $\tilde{r_0} = Q^{-1} r_0$;
set $p_0 = A^t Q^{-t} \tilde{r_0}$;
for $i = 0$ **step** 1 until convergence **do**
begin

$$a_i = \frac{(\tilde{r_i}, \tilde{r_i})}{(p_i, p_i)};$$
$$y_{i+1} = y_i + a_i p_i;$$
$$\tilde{r}_{i+1} = \tilde{r_i} - a_i Q^{-1} A p_i;$$
$$b_i = \frac{(\tilde{r}_{i+1}, \tilde{r}_{i+1})}{(\tilde{r_i}, \tilde{r_i})};$$
$$p_{i+1} = A^t Q^{-t} \tilde{r}_{i+1} + b_i p_i;$$

end

Here (x, y) denotes the inner product of x and y. For this algorithm, a minimum of $5(n + 1)$ storage locations is required (in addition to that for A). The vectors y, $\tilde{r}$, and p all require their own locations; $Q^{-t}\tilde{r}$ can share with Ap; $Q^{-1}Ap$ can share with $A^t Q^{-t}\tilde{r}$. The computational cost per iteration of this algorithm is:

(a) two preconditioning solves ($Q^{-1}v$ and $Q^{-t}v$);

(b) two matrix-vector products (Av and $A^t v$);

(c) $5(n + 1)$ multiplications (the inner products (p, p) and $(\tilde{r}, \tilde{r})$, ap, bp, and $aQ^{-1}Ap$).

3.1. Alternatives for solving (10). There are three main approaches to solving (10):

(1) In the block factorization approach to the problem, a block elimination algorithm is used instead of working with the whole matrix A directly. Such an algorithm would take advantage of the special properties of the submatrix B.

(2) The general approach works directly with the whole matrix A without taking any special advantage of the fact that the submatrix B contained in A is symmetric.

(3) The splitting approaches lie somewhere between (1) and (2). Here A is split into the sum of a symmetric matrix M and a low rank correction L. These methods also take advantage of the fact that the leading submatrix B is symmetric and can use conjugate gradient algorithms requiring a symmetric coefficient matrix.

APPROACH 1 (block factorization methods). The linear system (10) can also be written as

$$Ay = \begin{pmatrix} B & f \\ c^t & d \end{pmatrix} \begin{pmatrix} y' \\ y'' \end{pmatrix} = \begin{pmatrix} b' \\ b'' \end{pmatrix}.$$

A block-elimination algorithm [5] would be:

factor B;
solve $Bv = f$;
solve $Bw = b'$;
compute $y'' = (b'' - c^t w)/(d - c^t v)$;
compute $y' = w - y''v$.

With such block factorization methods, the work consists mainly of one factorization of B (assuming that is possible) and two backsolves with the factors of B. Observe that block elimination will frequently fail in the homotopy context, because even though rank $A = n + 1$ and rank $(B \quad f)$ = rank $D\rho_a = n$, it may very well happen that $B = D_x\rho_a$ is singular (rank $B = n - 1$). Singular B can be handled by deflation techniques [5]–[8], resulting in a direct algorithm very similar to other direct algorithms discussed below under matrix splittings. If the deflated systems were solved iteratively, this would constitute yet another iterative algorithm with no apparent advantage over the other iterative algorithms considered here. Deflation and block elimination will not be considered further.

APPROACH 2 (general methods). These algorithms work on the nonsymmetric A directly. If y_0 is an initial approximation of y, and r_0 the corresponding residual vector $r_0 = b - Ay_0$, then the Krylov subspace methods consist of finding an approximate solution belonging to the affine subspace $y_0 + K_j$, where K_j is the Krylov subspace generated by $r_0, Ar_0, \cdots A^{j-1}r_0$. There are several such methods besides Craig's method known as Orthomin(k) [44], Orthodir and Orthores [59], the Incomplete Orthogonalization Method [40], the GCR method [18], and the GMRES method [42]. Typical of

these methods is the preconditioned Orthomin(k) algorithm, given by:

choose y_0;
set $r_0 = b - Ay_0$;
set $\tilde{r_0} = Q^{-1}r_0$;
set $p_0 = r_0$;
for $i = 0$ **step** 1 until convergence **do**
begin

$$a_i = \frac{(\tilde{r_i}, Q^{-1}Ap_i)}{(Q^{-1}Ap_i, Q^{-1}Ap_i)};$$

$$y_{i+1} = y_i + a_i p_i;$$

$$\tilde{r}_{i+1} = \tilde{r}_i - a_i Q^{-1}Ap_i;$$

$$b_j^{(i)} = -\frac{(Q^{-1}A\tilde{r}_{i+1}, Q^{-1}Ap_j)}{(Q^{-1}Ap_j, Q^{-1}Ap_j)}, \qquad j = \max\{0, i-k+1\}, \cdots i;$$

$$p_{i+1} = \tilde{r}_{i+1} + \sum_{j=(i-k+1)_+}^{i} b_j^i p_j;$$

end

where $(i-k+1)_+ \equiv \max\{0, i-k+1\}$. As for the storage costs, Ap_j is overwritten by $Q^{-1}Ap_j$ and $A\tilde{r}$ by $Q^{-1}A\tilde{r}$. Thus storage is required for y, $\tilde{r}$, $\{p_j\}_{(i-k+1)_+}^{i}$, $\{Q^{-1}Ap_j\}_{(i-k+1)_+}^{i}$, and $Q^{-1}A\tilde{r}$.

However, Orthomin(k) is guaranteed to converge only for positive definite coefficient matrices A (equivalently, A with positive definite symmetric part $(A + A^t)/2$). (Some authors define *positive definite* only for symmetric matrices, while others say A is positive definite if $x^tAx > 0$ for all $x \neq 0$ in E^n, whether A is symmetric or not. This latter meaning is used here.) More general systems $Ax = b$, where A is not positive definite, can be solved by applying Orthomin(k) to the transformed system $ZAx = Zb$, where Z is nonsingular and ZA is positive definite. The matrix Z must be known and used explicitly in the iteration, a major obstacle to the general applicability of Orthomin(k).

The GCR method may also break down if the coefficient matrix is not positive definite. Although Orthodir does not break down in this case, it is observed to have stability problems [40]. GMRES, on the other hand, although equivalent to GCR for positive definite coefficient matrices, can be used to solve systems for which the coefficient matrix is not positive definite, and requires half as much storage as GCR. However, GMRES requires storage of the order of the number of iterations performed for convergence. Hence, the algorithm is used iteratively, i.e., it is restarted every k steps, where k is a fixed parameter, leading to the following GMRES(k) algorithm [42]:

choose y_0, tol;
set $r_0 = b - Ay_0$;
while $\|r_0\| >$ tol **do**

begin

set $v_1 = r_0/\|r_0\|$;

for $j = 1$ **step** 1 **until** k **do**

begin

for $i = 1$ **step** 1 **until** j **do** $h_{i,j} = (Av_j, v_i)$;

$$\tilde{v}_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j} v_i;$$

$$h_{j+1,j} = \|\tilde{v}_{j+1}\|;$$

$$v_{j+1} = \tilde{v}_{j+1} \,/\, h_{j+1,j}$$

end

Solve $\min_x \left\| \|r_0\| e_1 - \bar{H}_k x \right\|$ for x_k where $\bar{H}_k$ is described in [42];

set $y_0 = y_0 + V_k x_k$; **set** $r_0 = b - Ay_0$

end

In practice the algorithm calculates $\|r_j\|$ ($\|r_j\|$ can be calculated without forming y_j or $r_j = b - Ay_j$ explicitly) at each iteration of the j loop, and breaks the j loop if $\|r_j\| < \text{tol}$ [42], [45]. Also, it is important in practice that the classical Gram–Schmidt process in the inner j loop be replaced by the modified Gram–Schmidt process to ensure stability. GMRES(k), like Orthomin(k), is guaranteed to converge when the coefficient matrix is positive definite. However, for an indefinite coefficient matrix, GMRES(k), while it does not break down, may fail because the residual norms at each step, although nonincreasing, do not converge to zero.

APPROACH 3 (coefficient matrix splittings). There are several ways of splitting the coefficient matrix A in (10) as the sum of a symmetric matrix M and a low rank matrix L. The choice (c^t, d) as the last row of M gives the splitting

$$M = \begin{pmatrix} B & c \\ c^t & d \end{pmatrix}, \qquad L = ue_{n+1}^t, \qquad u = \begin{pmatrix} f - c \\ 0 \end{pmatrix}, \tag{11}$$

where e_{n+1} is a vector with 1 in the $(n+1)$st component and zeros elsewhere. There are many reasonable choices for (c^t, d), discussed later (recall that (c^t, d) can be almost any, in the sense of Lebesgue measure, vector for which (10) produces a solution to the true problem (9) or $[D\rho_a]y = 0$). The linear system $Ay = b$ is then solved by applying iterative techniques to two linear systems with coefficient matrix M followed by the Sherman–Morrison formula; the algorithmic details of this are in the next section. Another possibility would be to compute a symmetric indefinite factorization of M, and not use iterative methods at all. However, this destroys the skyline data structure containing M, and a tacit assumption here is that the skyline data structures must be preserved. If it were acceptable to destroy the skyline data structure, this direct approach would likely be the most efficient of all for skyline sparsity patterns, but would not generalize to arbitrary sparsity patterns (which the iterative methods will).

Another way of splitting up the coefficient matrix A is

$$A = D - A_L - A_U, \tag{12}$$

where D is the diagonal of A, A_L is the strict lower triangle of $-A$, and A_U is the strict upper triangle of $-A$. The symmetric successive over-relaxation (SSOR) iterative method [57] is the following two stage algorithm:

$$(D - \omega A_L)x_{i+1/2} = [(1-\omega)D + \omega A_U]x_i + \omega b,$$

$$(D - \omega A_U)x_{i+1} = [(1-\omega)D + \omega A_L]x_{i+1/2} + \omega b,$$

where ω is a real scalar parameter between 0 and 2. With

$$Q = \frac{1}{\omega(2-\omega)}(D - \omega A_L)D^{-1}(D - \omega A_U),$$

this method can be formulated as a one step algorithm

$$Qx_{i+1} = (Q - A)x_i + b.$$

In the homotopy context, D^{-1} frequently does not exist, and a diagonal matrix Σ such that $[\text{diag}\,(A + \Sigma)]^{-1}$ does exist may not be of low rank (meaning that the solution for A cannot be easily recovered from the solution for $A + \Sigma$). Consequently, SSOR and methods based on similar splittings are of limited utility in the homotopy context; in fact, SSOR failed for all the test problems in §5. A few experiments were also tried with SSOR ($\omega = 1$) as a preconditioner, but it was not competitive, and is not considered further here.

3.2. Some preconditioning techniques. This section considers some preconditioning techniques to be used in conjunction with the algorithms just described. Preconditioning matrices constructed from approximate factorizations of the coefficient matrix are considered first. A lower triangular matrix L and an upper triangular matrix U that are in some sense approximations of the factors in the LU factorization of A, but that are also sparse, are constructed. The preconditioning matrix is the product $Q = LU$. The heuristic used to insure that the preconditioning is inexpensive to implement is to force the factors to be sparse by allowing nonzeros only within a specified set of locations.

(i) *The incomplete* LU *factorization* (ILU). Let Z be a set of indices contained in $\{(i,j) \mid 1 \le i,j \le N,\ i \ne j\}$, typically where A is known to be zero. The incomplete LU factorization is given by $Q = LU$, where L and U are lower triangular and unit upper triangular matrices, respectively, that satisfy

$$\begin{cases} L_{ij} = U_{ij} = 0, & (i,j) \in Z, \\ Q_{ij} = A_{ij}, & (i,j) \notin Z. \end{cases}$$

The incomplete LU factorization algorithm is:

for $i = 1$ **step** 1 **until** N **do**
  for $j = 1$ **step** 1 **until** N **do**
    if $((i,j) \notin Z)$ **then**
      begin

$$s_{ij} = A_{ij} - \sum_{t=1}^{\min\{i,j\}-1} L_{it}U_{tj};$$

      if $(i \ge j)$ **then** $L_{ij} = s_{ij}$ **else** $U_{ij} = s_{ij}/L_{ii}$;
      end

It can happen that L_{ii} is zero in this algorithm. In this case L_{ii} is set to a small positive number, so that $Q_{ii} \neq A_{ii}$.

(ii) *The modified incomplete* LU *factorization* (MILU). Let Z be the set of indices that determine the zero structure, and assume that $(i,i) \notin Z$, $1 \leq i \leq N$. The modified incomplete LU factorization is given by $Q = LU$, where L and U are lower triangular and unit upper triangular matrices, respectively, that satisfy

$$\begin{cases} L_{ij} = U_{ij} = 0, & (i,j) \in Z, \\ Q_{ij} = A_{ij}, & (i,j) \notin Z, i \neq j, \\ \sum_{j=1}^{N} (Q_{ij} - A_{ij}) = \alpha, & 1 \leq i \leq N, \end{cases}$$

where α is a scalar. The modified incomplete LU factorization algorithm is:

for $i = 1$ **step** 1 **until** N **do**

begin

$L_{ii} = \alpha$;

for $j = 1$ **step** 1 **until** N **do**

begin

$$s_{ij} = A_{ij} - \sum_{t=1}^{\min\{i,j\}-1} L_{it} U_{tj};$$

if $((i,j) \notin Z)$ **then**

begin

if $(i > j)$ **then** $L_{ij} = s_{ij}$;

if $(i = j)$ **then** $L_{ii} = L_{ii} + s_{ii}$;

if $(i < j)$ **then** $\tilde{U}_{ij} = s_{ij}$;

end

else $L_{ii} = L_{ii} + s_{ij}$;

end

for $j = i + 1$ **step** 1 **until** n **do**

$U_{ij} = \tilde{U}_{ij} / L_{ii}$;

end

Since LU factorizations preserve a skyline sparsity structure, the MILU factorization is the same as the ILU factorization for $\alpha = 0$. The motivation for the MILU factorization is to control the elements of Q where it does not match A, at least in an average sense. In the homotopy context here with skyline A and $\alpha > 0$, Q can be construed as an approximation to A that is closer to (or more) positive definite than A.

4. Algorithms for computing $\ker[D\rho_a]$. As discussed in §2 for the normal flow algorithm, a corrector step ΔZ is the unique minimum norm solution of (9), which uses the solution of the rectangular linear system $[D\rho_a]y = 0$. This section describes various algorithms for the solution of such linear systems.

Let $(\bar{x}, \bar{\lambda})$ be a point on the zero curve γ, and $\bar{y}$ the unit tangent vector to γ at $(\bar{x}, \bar{\lambda})$ in the direction of increasing arc length s. Then the matrix

$$A = \begin{pmatrix} D_x\rho_a(x,\lambda) & D_\lambda\rho_a(x,\lambda) \\ c^t & d \end{pmatrix}, \tag{13}$$

where $(c^t \quad d)$ is any vector outside a set of measure zero (a hyperplane), is invertible at $(\bar{x}, \bar{\lambda})$ and in a neighborhood of $(\bar{x}, \bar{\lambda})$. Thus the kernel of $D\rho_a$ can be found by solving the linear system of equations

$$Ay = \alpha e_{n+1} = b, \tag{14}$$

where $(c^t \quad d)\bar{y} = \alpha$.

The coefficient matrix A in the linear system of equations (14) has a very special structure which can be exploited in several ways. Note that the leading $n \times n$ submatrix of A is $D_x\rho_a$, which is symmetric and sparse, but possibly indefinite. Since symmetry is advantageous for some algorithms, A can be made symmetric and invertible by choosing $c = D_\lambda\rho_a$. If rank $D_x\rho_a = n - 1$, then $D_\lambda\rho_a$ is not a linear combination of the columns of $D_x\rho_a$, because rank $\left[D_x\rho_a \; D_\lambda\rho_a\right] = n$ by the homotopy theory. Thus $c^t = \left(D_\lambda\rho_a\right)^t$ is not a linear combination of the rows of the symmetric matrix $D_x\rho_a$, and the

$$\text{row rank} \begin{bmatrix} D_x\rho_a \\ \left(D_\lambda\rho_a\right)^t \end{bmatrix} = n. \tag{15}$$

Finally,

$$\begin{pmatrix} D_\lambda\rho_a \\ * \end{pmatrix}$$

is not a linear combination of the first n columns of A, so the column rank $A = n + 1$ for *any* choice of d. Now suppose that rank $D_x\rho_a = n$. Then

$$\text{rank} \begin{bmatrix} D_x\rho_a \\ \left(D_\lambda\rho_a\right)^t \end{bmatrix} = n, \tag{16}$$

and it suffices to choose d to make the last column of A independent from the first n columns. $D_\lambda\rho_a$ is a unique linear combination of the columns of $D_x\rho_a$, and any choice of d other than this combination of the components of $\left(D_\lambda\rho_a\right)^t$ will make the $(n+1)$st column independent. Let $\bar{A}$ denote A at $(\bar{x}, \bar{\lambda})$. Since $\dim[\ker(\bar{A})] \leq 1$, $\bar{A}y = 0$ implies $y = \alpha\bar{y}$, and thus with $\bar{y}^t = \left(\hat{y}^t, \bar{y}_{n+1}\right)$, $\left(D_\lambda\rho_a(\bar{x}, \bar{\lambda})\right)^t\hat{y} + d\,\bar{y}_{n+1} = 0$. Choosing any $\beta \neq 0$ and solving $\left(D_\lambda\rho_a(\bar{x}, \bar{\lambda})\right)^t\hat{y} + d\,\bar{y}_{n+1} = \beta$ for d ($\bar{y}_{n+1} \neq 0$ since rank $D_x\rho_a(\bar{x}, \bar{\lambda}) = n$) gives a d such that $\text{rank}(A) = n + 1$ for (x, λ) near $(\bar{x}, \bar{\lambda})$.

Observe also that if $D_x\rho_a$ is positive definite, choosing $d > 0$ sufficiently large guarantees that

$$A = \begin{pmatrix} D_x\rho_a & D_\lambda\rho_a \\ (D_\lambda\rho_a)^t & d \end{pmatrix}$$

is also positive definite.

Proof. Since A is symmetric, by Sylvester's Theorem A is positive definite if and only if all its leading principal minors are positive. Since $D_x\rho_a$ is positive definite, the first n leading principal minors are positive, and it suffices to show $\det A > 0$. Expanding $\det A$ along the last column,

$$\det A = d \cdot \det D_x\rho_a + \text{terms not involving } d > 0$$

for $d > 0$ sufficiently large. □

Another approach is to attack (14) indirectly as follows. Write

$$A = M + L, \tag{17}$$

where

$$M = \begin{pmatrix} D_x\rho_a(\bar{x}, \bar{\lambda}) & c \\ c^t & d \end{pmatrix}, \tag{18}$$

$$L = ue_{n+1}^t, \qquad u = \begin{pmatrix} D_\lambda\rho_a(\bar{x}, \bar{\lambda}) - c \\ 0 \end{pmatrix}.$$

Observe that for almost all choices of $(c^t \quad d)$ the symmetric part M is also invertible. Then using the Sherman–Morrison formula, the solution y to the original system $Ay = b$ can be obtained from

$$y = \left[I - \frac{M^{-1}ue_{n+1}^t}{(M^{-1}u)^t e_{n+1} + 1} \right] M^{-1}b, \tag{19}$$

which requires the solution of two linear systems $Mz = u$ and $Mz = b$ with the sparse, symmetric, invertible matrix M. The scheme (17)–(19) was proposed in [27], and further investigated by Chan and Saad [9]. First, the HOMPACK approach to the solution of these linear systems will be discussed.

Let $|\bar{y}_k| = \max_i |\bar{y}_i|$ define the index k. In HOMPACK, $(c^t \quad d) = e_k^t$, where e_k is a vector with 1 in the kth component and zeros elsewhere. Hence (13) becomes

$$A = \begin{pmatrix} D\rho_a(x, \lambda) \\ e_k^t \end{pmatrix}.$$

The kernel of $D\rho_a$ can be found by solving the linear system of equations

$$Ay = \bar{y}_k e_{n+1} = b.$$

Again, splitting the coefficient matrix as

$$A = M + L$$

gives a symmetric

$$M = \begin{pmatrix} D_x\rho_a(\bar{x},\bar{\lambda}) & * \\ e_k^t & \end{pmatrix},$$

$$L = ue_{n+1}^t, \qquad u = \begin{pmatrix} D_\lambda\rho_a(\bar{x},\bar{\lambda}) \\ 0 \end{pmatrix} - e_k(1-\delta_{k,n+1}).$$

(The Kronecker $\delta_{k,n+1}$ takes care of the special case $k = n+1$.) Then the Sherman–Morrison formula (19) is used for the solution y of the linear system (14). Craig's preconditioned algorithm is used for solving the systems $Mz = u$ and $Mz = b$. The preconditioning matrix Q is taken as a positive definite approximation of M (the *Gill–Murray preconditioner*, described in detail in [21] and [52]). The details are intricate, but essentially Q is computed as a Cholesky factorization of $M + \Sigma$, where Σ is a positive semidefinite diagonal matrix chosen to guarantee that Q is well conditioned. This algorithm is not especially well matched to the skyline data structure.

If M had only one or two negative eigenvalues, then after several rank one updates making M positive definite, a direct Cholesky factorization could be obtained, and then the solution to (14) recovered after several more applications of (19). This direct algorithm for solving (14) would be effective for such M, but since only one or two negative eigenvalues for M cannot be assumed in general (the M for a large shallow dome problem has many negative eigenvalues along the unloading portions of the equilibrium curve), a direct rank one update/Cholesky scheme would not be suitable for HOMPACK.

There are several other schemes which could be used instead of the one in HOMPACK for finding the kernel of $D\rho_a$, for example,

(i) using different last rows for the augmented coefficient matrix A of (13), i.e., other vectors $(c^t \quad d)$ instead of e_k^t;

(ii) using other preconditioners on M;

(iii) using other algorithms for the solution of the linear systems $Mz = u$ and $Mz = b$, e.g., Orthomin(k), SSOR, etc., instead of Craig's algorithm;

(iv) doing (i), (ii), or (iii) on the nonsymmetric A directly instead of on the symmetric M in the splitting $A = M + L$.

Combining the preconditioning techniques with the algorithms for solving linear systems with different last rows for A produces a large number of possible methods. The next section focuses on a subset of these possible methods and compares their efficiency.

5. Numerical experiments. Of the various algorithmic possibilities mentioned in the previous section, those considered further are given short names in the list below. Some possibilities do not make sense or are impractical in the homotopy context, and thus are not considered. Of the almost all mathematically valid choices for the last row $(c^t \quad d)$ of A, only e_k^t (the easiest to implement), $c = D_\lambda\rho_a(\bar{x},\bar{\lambda})$ (the easiest symmetrization of A), and the tangent vector $\bar{y}^t$ at the previous point on the zero curve (the optimal choice for conditioning, since it is orthogonal to the top n rows of A at $(\bar{x},\bar{\lambda})$) are used.

SC – $A = M + L$ splitting, Craig's method with M, no preconditioning;
SCGM – $A = M+L$ splitting, Craig's method with M, Gill–Murray preconditioning from HOMPACK;
SCILU – $A = M + L$ splitting, Craig's method with M, incomplete LU preconditioning;
SCMILU – $A = M + L$ splitting, Craig's method with M, modified incomplete LU preconditioning;
C – no splitting, Craig's method with A, no preconditioning;
CGM – no splitting, Craig's method with A, Gill–Murray preconditioning from HOMPACK;
CILU – no splitting, Craig's method with A, incomplete LU preconditioning;
CMILU – no splitting, Craig's method with A, modified incomplete LU preconditioning.
SR – $A = M + L$ splitting, GMRES(2) with M, no preconditioning;
SRGM – $A = M + L$ splitting, GMRES(2) with M, Gill–Murray preconditioning from HOMPACK;
SRILU – $A = M+L$ splitting, GMRES(2) with M, incomplete LU preconditioning;
SRMILU – $A = M + L$ splitting, GMRES(2) with M, modified incomplete LU preconditioning;
R – no splitting, GMRES(2) with A, no preconditioning;
RGM – no splitting, GMRES(2) with A, Gill–Murray preconditioning from HOMPACK;
RILU – no splitting, GMRES(2) with A, incomplete LU preconditioning;
RMILU – no splitting, GMRES(2) with A, modified incomplete LU preconditioning.

The test problems are now described in detail, beginning with the shallow arch structural response problem.

5.1. Shallow arch. The equations of equilibrium of the arch are obtained from the principle of the stationary value of the total potential energy, according to which, of all the kinematically admissible displacement fields, the one that makes the total potential energy of a structure stationary also satisfies its equations of equilibrium. The total potential energy π of a structure is given by the sum of its strain energy and the potential of external loads.

The shallow arch of Fig. 1 is discretized by an assemblage of straight p-q frame elements such as those described in [26]. A frame element is a structural component that is initially straight and undergoes axial, bending, and torsional deformation resulting from finite displacements and rotations of its ends (nodes) p and q. The displacements of the end q relative to the end p are

$$\begin{pmatrix} \delta u \\ \delta v \\ \delta w \end{pmatrix} = [T]_p \begin{pmatrix} X_q - X_p \\ Y_q - Y_p \\ Z_q - Z_p \end{pmatrix} - \begin{pmatrix} L \\ 0 \\ 0 \end{pmatrix} + [T]_p \begin{pmatrix} U_q - U_p \\ V_q - V_p \\ W_q - W_p \end{pmatrix},$$

where L is the initial rigid body length, and U_i, V_i, W_i ($i = p$ or q) denote the global displacements of the nodes. The matrix $[T]_p$ can be shown to be [26] $[T]_p = \left[T_1(\phi_x, \phi_y, \phi_z)\right]\left[T_1(\theta_{xp}, \theta_{yp}, \theta_{zp})\right]$ with

$$\left[T_1(\alpha_x, \alpha_y, \alpha_z)\right] = \begin{pmatrix} c_y c_z & c_y s_z & -s_y \\ -c_x s_z + s_x s_y c_z & c_x c_z + s_x s_y s_z & s_x c_y \\ s_x s_z + c_x s_y c_z & -s_x c_z + c_x s_y s_z & c_x c_y \end{pmatrix},$$

$c_i = \cos\alpha_i$ and $s_i = \sin\alpha_i$ for $i = x$, y, and z. Angles ϕ_x, ϕ_y, and ϕ_z are the initial orientation angles and angles θ_{xp}, θ_{yp}, and θ_{zp} are the rigid body rotations of the end p. In the equation for $[T]_p$, Euler angle transformations are implied with the order of the rotations being α_z, α_y, and α_x.

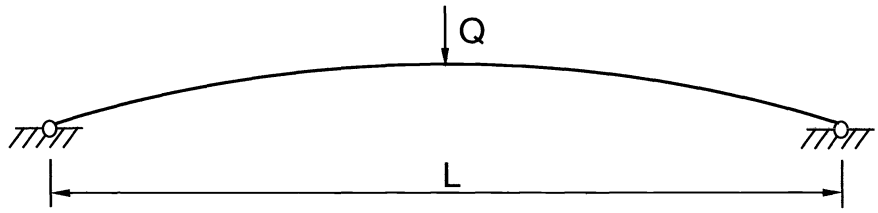


FIG. 1. *Shallow arch.*

Similarly, with the restriction of small relative rotations within the element, the rotations ψ_x, ψ_y, ψ_z of the end q relative to the end p are

$$\begin{pmatrix} \psi_x \\ \psi_y \\ \psi_z \end{pmatrix} = [T]_p \begin{pmatrix} \theta_{xq} - \theta_{xp} \\ \theta_{yq} - \theta_{yp} \\ \theta_{zq} - \theta_{zp} \end{pmatrix}.$$

With the relative generalized displacements $(\delta u, \delta v, \delta w)$ and (ψ_x, ψ_y, ψ_z) known, the usual deformation patterns of the reference axis of the beam element in the corotational coordinate system are assumed to be

$$u(\xi) = \xi\frac{\delta u}{L}, \qquad v(\xi) = \frac{1}{L}(3\xi^2 - 2\xi^3)(\delta v - z_s\psi_x) + (\xi^3 - \xi^2)\psi_z,$$
$$\beta = \xi\psi_x, \qquad w(\xi) = \frac{1}{L}(3\xi^2 - 2\xi^3)(\delta w + y_s\psi_x) - (\xi^3 - \xi^2)\psi_y,$$

where $\xi = x/L$ and y_s and z_s are the coordinates of the shear center of the cross section of the beam. The strain at any point (y, z) on the cross-section of the frame element can be shown to be

$$\epsilon = \frac{\delta u}{L} - \eta\left[\frac{6}{L}(1 - 2\xi)(\delta v - z_s\psi_x) + 2(3\xi - 1)\psi_z\right] - \zeta\left[\frac{6}{L}(1 - 2\xi)(\delta w + y_s\psi_x) - 2(3\xi - 1)\psi_y\right],$$

with $\eta = y/L$ and $\zeta = z/L$. In these equations it is implicitly assumed that the lateral displacements and twists are referenced to a longitudinal axis through the shear center, while the axial displacements and rotations are referenced to the centroidal axis.

The total potential energy of such a discretized model of the arch can be expressed as

$$\pi = \sum_{e=1}^{m} U^e - q^t Q,$$

where U^e is the strain energy of the eth element, $e = 1, \cdots, m$, q is the vector of nodal displacement degrees of freedom of the entire model and Q is the vector of externally applied loads. The strain energy U^e of the eth frame element is given by

$$U^e = \frac{E}{2}\int_V \epsilon^2 dv = \frac{E}{2}\int_0^{L_e}\int_{A_e} \epsilon^2 dA\, dx,$$

where ϵ is the strain of a point (x, y, z) of the beam, which was derived above. Substituting for ϵ and doing the integration gives

$$U^e = U_{\text{p-q}} = \frac{E}{2L_e}\left\{A_e(\delta u)^2 + \frac{12}{L_e^2}I_z\left[(\delta v)^2 + \frac{1}{3}L_e^2\psi_z^2 - L_e\delta v\,\psi_z\right] + \frac{12}{L_e^2}I_y\left[(\delta w)^2 + \frac{1}{3}L_e^2\psi_y^2 + L_e\delta w\,\psi_y\right]\right\},$$

where A_e is the cross-sectional area, and I_y and I_z are the cross-sectional moments of inertia about the y and z axes, respectively. It is evident that the potential energy π of the model is a highly nonlinear function of the nodal displacements. The equations of equilibrium of the model are obtained by setting the variation $\delta\pi$ to zero, or equivalently by

$$\nabla\pi = 0.$$

Closed form analytical expressions for $\nabla\pi$ can be obtained with some difficulty, but obtaining the Jacobian matrix of $\nabla\pi$ analytically seems out of the question. Hence the Jacobian matrix of the equilibrium equations is obtained by finite difference approximations.

By symmetry only half the arch need be modelled, and the results here are for the arch parameters used in [28], with a full arch load of 3000 lbs. This is just below the limit point. To go through the limit point and along the unloading portion of the equilibrium curve apparently requires very accurate Jacobian matrices and numerical linear algebra, and none of these iterative linear equation solvers used in HOMPACK were able to go past the limit point without tweaking the HOMPACK step size control parameters.

5.2. Shallow dome. The shallow dome of Fig. 2 is built up from space truss elements with three global displacement degrees of freedom (u_1, u_2, u_3) at each of the two nodes. For an element of original length L between its two nodes p and q, the change in length δL is given by

$$\delta L = \left[\sum_{i=1}^{3}(x_{qi} + u_{qi} - x_{pi} - u_{pi})^2\right]^{1/2} - \left[\sum_{i=1}^{3}(x_{qi} - x_{pi})^2\right]^{1/2},$$

where x_{ij}, u_{ij}, $i = p, q$; $j = 1, 2, 3$ are the global coordinates and displacements of the two nodes. This can be simplified to

$$\delta L = L\left[1 + \sum_{i=1}^{3}\left(\frac{2(\Delta x_i \Delta u_i)}{L^2} + \frac{(\Delta u_i)^2}{L^2}\right)\right]^{1/2} - L,$$

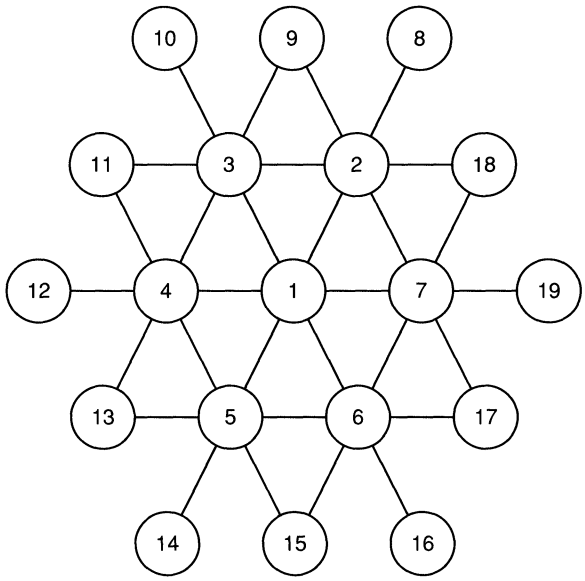


FIG. 2. *Triangulation for* 21 *degree of freedom shallow dome.*

where Δ is the difference operator for the q and p values. Accordingly, the axial strain in the eth element is

$$\epsilon^e = \frac{\delta L}{L} = \left[1 + \sum_{i=1}^{3} \left(\frac{2(\Delta x_i \Delta u_i)}{L^2} + \frac{(\Delta u_i)^2}{L^2} \right) \right]^{1/2} - 1.$$

The strain energy of the eth element in a purely linearly elastic response is given by

$$U_s^e = \frac{E}{2} \int_V (\epsilon^e)^2 dV = \frac{EA^e L^e}{2} (\epsilon^e)^2,$$

where E and A are the Young's modulus and cross-sectional area, respectively, of the eth element.

The total potential energy of the dome is then given by

$$\pi = \sum_{e=1}^{m} U_s^e - U^T Q,$$

where U_i, $i = 1, \cdots, 6$ are the six components u_{qk}, u_{pk}, $k = 1, 2, 3$, and Q is the generalized force vector. The equations of equilibrium of the model are then obtained by setting

$$\nabla \pi = \sum_{e=1}^{m} EA^e L^e \nabla \epsilon^e - Q = 0.$$

Both the gradient of π as well as its Hessian can be evaluated explicitly without resorting to finite differencing operations as in the case of the frame element used to model the shallow arch.

The effect of modelling the shallow dome with truss elements in concentric rings is that changing the number of truss elements changes the model and its behavior. Thus the dome problems with different degrees of freedom reported in the tables are qualitatively different, with different buckling loads and bifurcation points. The results reported here are for shallow domes with base radius 720 and sphere radius 3060, and a point load at the very top.

5.3. Artificial turning point problem. The turning point problem is derived from the system of equations

$$F(\mathbf{x}) = \big(F_1(\mathbf{x}), F_2(\mathbf{x}), \cdots, F_N(\mathbf{x})\big)^t = 0$$

where

$$F_i(\mathbf{x}) = \tan^{-1}\big(\sin[x_i(i \bmod 100)]\big) - \frac{(x_{i-1} + x_i + x_{i+1})}{20}, \qquad i = 1, \cdots, N,$$

and $x_0 = x_{N+1} = 0$. The zero curve γ tracked from $\lambda = 0$ to $\lambda = 1$ corresponds to $\rho_a(x, \lambda) = (1 - .8\lambda)(x - a) + .8\lambda F(x)$, where a was chosen artificially to produce turning points in γ. HOMPACK had no difficulty going through numerous turning points using iterative linear equation solvers.

Tables 1–6 show some timing results for these three test problems. An asterisk indicates either that the iterative linear equation solver stalled, γ was lost because of inaccurate tangents from the linear equation solver, or the time was at least an order of magnitude larger than anything else in the table. The times are for tracking the entire zero curve γ and thus represent the solution of many linear systems of varying degrees of difficulty. The experiments were done in double precision using a single processor of a Sequent Symmetry S81 multiprocessor. The major headings are the acronyms for the algorithms, and the subheadings denote the choice $(c^t \quad d)$ for the last row of A. The MILU algorithms used $\alpha = 1$. There is asymmetry in the tables because some possibilities do not make sense. For instance, there is no CGM with e_k because the Gill–Murray preconditioner requires a symmetric matrix, and there are no S* with $D_\lambda \rho_a$ since the choice $c^t = \big(D_\lambda \rho_a\big)^t$ makes A symmetric so there is no need to split off a symmetric matrix M from A.

6. Discussion and conclusions. The convergence rate of conjugate gradient iterative methods for linear systems depends on the spectrum and the condition number of the coefficient matrix, and therefore one would predict $\bar{y}^t$ should be a better choice for the last row of A than e_k^t. Since $\bar{y}$ is orthogonal to the rows of $D\rho_a(\bar{x}, \bar{\lambda})$, a good approximation to the first n rows $D\rho_a(x, \lambda)$ of A, one expects A with $\bar{y}$ to be better conditioned than with e_k. Tables 1, 3, and 5 show that apparently this better conditioning does not compensate for the extra work involved in using $\bar{y}$. Although $\bar{y}$ is sometimes better than e_k, there seems to be no strong evidence that $\bar{y}$ is worth the trouble.

Figure 3 shows the condition numbers of A and $Q^{-1}A$ along γ for the shallow arch problem with $n = 29$ (CGM). The shallow arch problem is indeed a hard problem, but Fig. 3 alone would not suggest that—see the discussion of the shallow arch problem's spectra later. The Jacobian matrix $D_x \rho_a$ becomes indefinite near $\lambda = .88$, at which

TABLE 1
Execution time in seconds for shallow arch problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
29	1108	947	468	818	599	470	908	975
47	16904	17593	7322	10105	5674	5957	11538	12390
	SR		SRGM		SRILU		SRMILU	
29	*	*	461	*	559	443	*	*
47	*	*	5314	*	5796	6332	*	*

TABLE 2
Execution time in seconds for shallow arch problem.

	C			CILU			CMILU			CGM
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
29	856	884	919	533	458	443	841	845	900	464
47	14205	13591	14606	5794	5807	6776	9943	10968	10135	6921
	R			RILU			RMILU			RGM
29	*	*	*	443	431	506	*	*	*	429
47	*	*	*	5355	5304	5697	*	*	*	5260

TABLE 3
Execution time in seconds for shallow dome problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
21	57	86	108	57	21	25	92	141
546	3127	4803	2710	1787	492	630	4892	6687
1050	5615	8553	5107	3177	887	1133	8259	11672
	SR		SRGM		SRILU		SRMILU	
21	*	*	*	*	14	15	*	*
546	*	*	*	*	299	335	*	*
1050	*	*	*	*	559	625	*	*

TABLE 4
Execution time in seconds for shallow dome problem.

	C			CILU			CMILU			CGM
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
21	46	47	47	16	16	16	68	79	69	89
546	2495	2545	2573	355	369	365	3037	3585	3094	2233
1050	4504	4691	4690	632	665	651	5536	6327	5570	4313
	R			RILU			RMILU			RGM
21	*	*	*	11	12	11	*	*	*	*
546	*	*	*	230	241	232	*	*	*	*
1050	*	*	*	425	446	430	*	*	*	*

TABLE 5
Execution time in seconds for turning point problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
20	28	36	12	13	6	6	39	47
60	266	356	41	50	20	22	163	213
125	1635	2310	127	170	54	65	568	795
250	3026	3767	228	267	95	109	1032	1335
500	6279	7783	448	501	189	207	2130	2656
1000	14150	17768	1077	1174	434	490	4874	6052
	SR		SRGM		SRILU		SRMILU	
20	1301	*	13	18	4	4	120	*
60	*	*	47	*	13	14	*	*
125	*	*	*	*	36	40	*	*
250	*	*	*	*	60	69	*	*
500	*	*	*	*	119	131	*	*
1000	*	*	*	*	274	296	*	*

TABLE 6
Execution time in seconds for turning point problem.

	C			CILU			CMILU			CGM
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	$D_\lambda \rho_a$
20	17	19	21	4	7	4	24	26	27	5
60	167	176	186	13	22	13	109	112	118	22
125	1117	1132	1384	38	64	42	412	421	446	85
250	2296	1925	3873	66	110	74	765	699	726	134
500	4741	3899	8352	129	210	148	1573	1381	1465	260
1000	11577	9335	20375	323	493	353	3605	3031	3308	617
	R			RILU			RMILU			RGM
20	*	*	1263	3	3	3	43	70	75	6
60	*	*	*	9	10	9	*	*	2016	*
125	*	*	*	26	31	28	*	*	*	*
250	*	*	*	45	51	46	*	*	*	*
500	*	*	*	88	98	88	*	*	*	*
1000	*	*	*	199	225	202	*	*	*	*

point the Gill–Murray preconditioner ceases being nearly perfect. This figure is typical for the Gill–Murray preconditioner. (During the course of the experiments it was observed that points far from γ frequently generate much worse conditioned problems. This has important implications for curve tracking strategy, because large steps along γ will be offset by expensive numerical linear algebra to return to γ.)

Tables 2, 4, 6 show that there is no clear winner between e_k, $\bar{y}$, and $D_\lambda \rho_a$, and further that there is little correlation between the algorithm and the best choice for c. One is tempted to pick CGM with $c^t = \left(D_\lambda \rho_a\right)^t$ over SCGM with $c^t = e_k^t$, based on Tables 5, 6, and the fact that CGM only solves one linear system per tangent vector computation, as opposed to two linear systems with M for the splitting algorithms.

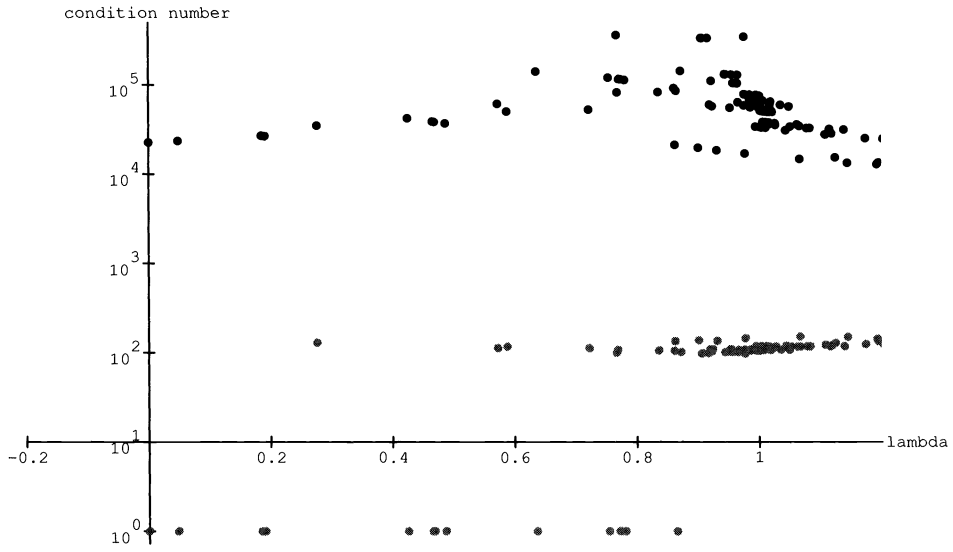


FIG. 3. *Condition numbers for* A (*black dots*) *and* $Q^{-1}A$ (*grey dots*) *against* λ *along* γ*; shallow arch,* $n = 29$, CGM.

However, from Tables 1 and 2, SCGM with e_k is substantially better than CGM with $D_\lambda \rho_a$. This demonstrates that only counting the number of linear system solves can be dangerously misleading. The results do indicate that for a given choice of c^t, when no preconditioning is used or when MILU preconditioning is used, it is slightly more efficient to use the no-splitting strategy than the splitting. However, with ILU preconditioning the differences between corresponding splitting and no-splitting cases are not at all significant.

Tables 1–6 seem to strongly support an argument for CILU as the best Craig method, even though the ILU factorization fails to exist at turning points, and is unstable whenever A is indefinite. What is not indicated in the tables, though, are all the homotopy curve tracking runs which failed because the ILU preconditioner failed to exist or generated an overflow, or the difficulty caused HOMPACK by inaccurate tangents resulting from the ILU. Because of this potential catastrophic failure or instability, the ILU preconditioner would never be seriously considered for robust homotopy algorithms. Still, the tables do show why numerical analysts' paranoia about unstable algorithms is not shared by engineers.

The algorithms SSOR and Orthomin(k), discussed earlier, are not shown in the tables because they totally fail at turning points and along unloading portions of equilibrium curves (for reasons stated in §3). When these methods do work, they can be very efficient (e.g., Orthomin(1) on A with $c^t = \left(D_\lambda \rho_a\right)^t$ took 443 (6092) seconds for the shallow arch problem with $n = 29$ (47)), but that is no consolation for homotopy curve tracking.

GMRES(k) has a solid theoretical justification [42], and has been used very successfully in a variety of contexts [4], [42], [45], [46]. Nevertheless, GMRES(k) with $k < n$ performed unacceptably on the test problems here without preconditioning.

For the shallow arch problem with $n = 29$ and tol $= 10^{-12}$, GMRES(29) on A with $c^t = \left(D_\lambda \rho_a\right)^t$ took 591 seconds, somewhat better than C or SC and comparable to CGM and SCGM. For $k = 1, 3, 25$, GMRES(k) took over a day of CPU time. Relaxing the tolerance to 10^{-6}, GMRES(25) took 18,330 seconds. This is especially noteworthy because the A matrices are symmetric and positive definite up to $\lambda = .88$, and mildly indefinite from there to $\lambda = 1$. For the turning point problem with $n = 20$, tol $= 10^{-12}$, $c^t = \left(D_\lambda \rho_a\right)^t$, the performance degradation from the full GMRES to GMRES(k) was dramatic. With $k = 20, 19, 18, 15, 10, 8$, GMRES($k$) took, respectively, 19, 117, 154, 375, 338, 420 seconds. Thus for these problems, without preconditioning, only the full GMRES method is competitive.

The tables also show results for GMRES(2), implemented with all the same preconditioners and choices of $(c^t \quad d)$ as was Craig's method. $k = 2$ was chosen because a preconditioned GMRES(2) requires exactly the same amount of storage as the preconditioned Craig's method, although, of course, the storage penalty for $k = 5$, say, is not significant. Numerous other runs were made with $k = 1, 3, 5$, or 10, but there was no substantial difference from $k = 2$ on the larger problems. In virtually all cases the asterisks in the tables correspond to a stalled residual norm somewhere along γ. It was noted, though, that many of the linear systems along γ were solved efficiently by GMRES(2). Perhaps the most disappointing failure was that of RGM on the shallow dome problem even for $n = 21$, because the Gill–Murray preconditioner was fairly good there. It is evident from the tables that GMRES(2), without nearly perfect preconditioning (ILU), is unsuitable for use in a general, robust homotopy curve tracking code like HOMPACK.

There are some theoretical results concerning the convergence of GMRES(k) given by Saad and Schultz [42]. These results give worst-case bounds on the rate of residual norm reduction which are determined by the distribution of eigenvalues of A. For the shallow arch and turning point problems, the eigenvalues of A were determined numerically along the homotopy curve, and the resulting bounds were often (although not in every case) found to guarantee only hopelessly slow residual norm reduction, indeed often to guarantee no residual norm reduction at all even when $k = n$.

Tables 7–12 show the average, maximum, and minimum number of conjugate gradient iterations per linear system solution along the homotopy zero curve γ for the same algorithms as Tables 1–6. Such iteration statistics give an intuitive feel for how the algorithms behave, and are sometimes very revealing. Tables 7 and 8 show that symmetry does improve the algorithms' efficiency, and that all other things being equal, achieving symmetric coefficient matrices is worthwhile. (The S* algorithms based on symmetry are not uniformly better, because all other things are not equal.) Note that in all cases for Craig's method the maximum number of conjugate gradient iterations is less than or equal to eight times the average, which says that the convergence behavior is fairly consistent. On the other hand the range between the minimum and maximum (for the C* algorithms) is as great as 3 to 536 (C for $n = 1000$ in Table 12), showing that there is a wide variation in the difficulty of the linear systems encountered along γ. The convergence behavior of GMRES(2) is not as consistent as for Craig's method, with the maximum being as much as 70 times the average (SRGM for $n = 47$ in Table 7).

TABLE 7

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for shallow arch problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
29	66,109,1	66,101,1	4,10,1	28,40,1	2,3,1	3,3,1	39,63,1	39,58,1
47	190,313,1	194,291,1	5,10,1	37,53,1	2,3,1	3,3,1	64,103,1	65,97,1
	SR		SRGM		SRILU		SRMILU	
29	*	*	4,92,1	*	1,2,1	1,2,1	*	*
47	*	*	2,140,1	*	1,2,1	1,2,1	*	*

TABLE 8

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for shallow arch problem.

	C			CILU		
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$
29	99,127,51	91,107,38	98,120,52	3,3,2	4,5,2	3,3,2
47	265,360,109	239,305,133	265,355,105	3,3,2	4,4,2	3,3,2
	CMILU			CGM		
29	56,68,30	56,65,35	55,65,30	—	—	6,7,2
47	87,119,48	91,102,53	87,131,48	—	—	6,7,2
	R			RILU		
29	*	*	*	1,1,1	1,2,1	1,1,1
47	*	*	*	1,1,1	1,2,1	1,1,1
	RMILU			RGM		
29	*	*	*	—	—	2,2,1
47	*	*	*	—	—	2,2,1

TABLE 9

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for shallow dome problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
21	17,31,1	24,36,1	16,115,1	7,46,1	2,3,1	2,3,1	14,30,1	21,32,1
546	38,75,1	54,87,1	15,113,1	9,63,1	2,3,1	3,2,1	24,45,1	37,71,1
1050	38,76,1	53,91,1	16,114,1	8,101,1	2,3,1	3,3,1	24,47,1	36,61,1
	SR		SRGM		SRILU		SRMILU	
21	*	*	*	*	1,2,1	1,2,1	*	*
546	*	*	*	*	1,2,1	1,2,1	*	*
1050	*	*	*	*	1,2,1	1,2,1	*	*

TABLE 10

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for shallow dome problem.

	C			CILU		
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$
21	26,36,14	26,36,14	26,36,14	2,3,2	2,3,2	2,3,2
546	58,81,17	57,82,17	58,82,18	2,3,2	2,3,2	2,3,2
1050	58,87,18	59,91,18	58,83,18	2,3,2	2,3,2	2,3,2
	CMILU			CGM		
21	19,25,8	22,28,8	19,24,8	—	—	23,113,2
546	34,47,12	38,52,12	34,45,11	—	—	22,111,2
1050	34,49,12	38,50,12	34,49,13	—	—	23,113,2
	R			RILU		
21	*	*	*	1,1,1	1,1,1	1,1,1
546	*	*	*	1,1,1	1,1,1	1,1,1
1050	*	*	*	1,1,1	1,1,1	1,1,1
	RMILU			RGM		
21	*	*	*	—	—	*
546	*	*	*	—	—	*
1050	*	*	*	—	—	*

TABLE 11

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for turning point problem.

	SC		SCGM		SCILU		SCMILU	
n	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$	e_k^t	$\bar{y}^t$
20	21,28,1	24,29,1	4,6,1	5,7,1	2,2,1	2,2,1	17,27,1	19,26,1
60	60,100,1	69,87,1	4,8,1	5,8,1	2,3,1	2,3,1	21,37,1	25,39,1
125	127,261,1	154,264,1	5,9,1	6,11,1	2,3,1	2,3,1	26,51,1	31,51,1
250	139,302,1	150,246,1	5,11,1	5,10,1	2,2,1	2,3,1	27,60,1	30,55,1
500	149,314,1	164,281,1	5,11,1	5,10,1	2,2,1	2,3,1	28,62,1	31,53,1
1000	151,312,1	162,289,1	5,11,1	5,11,1	2,2,1	2,3,1	28,64,1	31,56,1
	SR		SRGM		SRILU		SRMILU	
20	732,4446,1	*	6,58,1	8,75,1	1,1,1	1,1,1	49,315,1	*
60	*	*	6,54,1	*	1,1,1	1,1,1	*	*
125	*	*	*	*	1,1,1	1,1,1	*	*
250	*	*	*	*	1,1,1	1,1,1	*	*
500	*	*	*	*	1,1,1	1,1,1	*	*
1000	*	*	*	*	1,1,1	1,1,1	*	*

The Gill–Murray preconditioner is clearly excellent, as shown by the average number of iterations in Tables 7–12 and Fig. 3. It is more robust than the ILU and MILU preconditioners in the presence of turning points and indefinite $D_x \rho_a$. However, the shallow dome problem (Tables 3, 4, 9, and 10) shows that the Gill–Murray precon-

TABLE 12

Average, maximum, and minimum number of conjugate gradient iterations per linear system along homotopy curve for turning point problem.

	C			CILU		
n	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$	e_k^t	$\bar{y}^t$	$D_\lambda \rho_a$
20	24,29,1	24,28,1	26,31,1	2,2,1	4,5,1	2,3,1
60	70,86,1	69,84,1	74,91,2	2,3,1	4,7,1	2,3,2
125	159,292,1	151,232,1	179,328,3	2,3,1	4,5,1	2,3,2
250	196,404,1	150,246,1	231,407,3	2,3,1	4,5,1	2,3,2
500	216,427,1	165,337,1	268,489,3	2,3,1	4,6,1	2,3,2
1000	224,446,1	164,323,1	285,536,3	2,3,1	4,5,1	3,3,2
	CMILU			CGM		
20	20,23,1	20,23,1	20,26,3	—	—	3,9,2
60	26,36,1	25,34,1	26,36,2	—	—	3,12,1
125	33,49,1	31,48,1	33,53,3	—	—	5,15,2
250	36,61,1	30,45,1	32,48,1	—	—	4,15,2
500	38,74,1	31,53,1	33,53,3	—	—	5,16,2
1000	39,75,1	31,50,1	33,54,3	—	—	5,16,2
	R			RILU		
20	*	*	1905,21000,2	1,1,1	1,2,1	1,1,1
60	*	*	*	1,1,1	1,2,1	1,1,1
125	*	*	*	1,1,1	1,2,1	1,1,1
250	*	*	*	1,1,1	1,2,1	1,1,1
500	*	*	*	1,1,1	1,2,1	1,1,1
1000	*	*	*	1,1,1	1,2,1	1,1,1
	RMILU			RGM		
20	47,110,2	61,200,2	79,226,2	—	—	4,92,1
60	*	*	568,12486,2	—	—	*
125	*	*	*	—	—	*
250	*	*	*	—	—	*
500	*	*	*	—	—	*
1000	*	*	*	—	—	*

ditioner may do a very poor job indeed on strongly indefinite matrices (which occur on the unloading parts of the shallow dome equilibrium curve). While reducing the average number of iterations, the Gill–Murray preconditioner actually increases the maximum number of iterations compared to the unpreconditioned algorithm.

It would be possible to test separately each aspect of the iterative linear system solving algorithms, such as convergence rate, sensitivity to starting point, cost of preconditioning, storage cost, computational complexity per iteration, etc. What ultimately matters, however, is the combined performance of the total algorithm on a wide range of typical realistic problems. Measuring the performance along homotopy zero curves for nontrivial problems is an attempt to measure the overall performance *in situ*.

A succinct, albeit oversimplified summary of the discussion is that ILU preconditioning is the most efficient but it may completely fail for some cases, while the Gill–Murray preconditioner rarely fails but is somewhat slower, especially for very large or strongly indefinite problems. With somewhat imperfect preconditioning, Craig's method is more robust than GMRES(k) for $k \ll n$ for homotopy curve tracking.

Acknowledgments. The authors are indebted to the referees for excellent suggestions, and to Tony Chan, Dianne O'Leary, Philippe Toint, and David Young for comments on this work.

REFERENCES

[1] O. Axelsson, *Conjugate gradient type methods for unsymmetric and inconsistent systems of linear equations*, Linear Algebra Appl., 29 (1980), pp. 1–16.

[2] ———, *Solution of linear systems of equations: iterative methods*, in Sparse Matrix Techniques, Springer-Verlag, New York, 1976, pp. 1–51.

[3] O. Axelsson, S. Brinkkemper, and V.P. Ilin, *On some versions of incomplete block-matrix factorization iterative methods*, Linear Algebra Appl., 58 (1984), pp. 3–15.

[4] P. N. Brown and A. C. Hindmarsh, *Reduced storage matrix methods in stiff* ODE *systems*, J. Appl. Math. Comp., 31 (1989), pp. 40–91.

[5] T. F. Chan, *Deflation techniques and block-elimination algorithms for solving bordered singular systems*, Tech. Report 226, Department of Computer Science, Yale University, New Haven, CT, 1982.

[6] ———, *Deflated decomposition of solutions of nearly singular systems*, Tech. Report 225, Department of Computer Science, Yale University, New Haven, CT, 1982.

[7] ———, *On the existence and computation of* LU*-factorizations with small pivots*, Tech. Report 227, Department of Computer Science, Yale University, New Haven, CT, 1982.

[8] T. F. Chan and D. C. Resasco, *Generalized deflated block-elimination*, Tech. Report 337, Department of Computer Science, Yale University, New Haven, CT, 1985.

[9] T. F. Chan and Y. Saad, *Iterative methods for solving bordered systems with applications to continuation methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 438–451.

[10] S. N. Chow, J. Mallet-Paret, and J. A. Yorke, *Finding zeros of maps: Homotopy methods that are constructive with probability one*, Math. Comp., 32 (1978), pp. 887–899.

[11] P. Concus and G. H. Golub, *A generalised conjugate gradient method for nonsymmetric systems of linear equations*, in Lecture Notes in Economics and Mathematical Systems, 134, R. Glowinski and J. L. Lions, eds., Springer-Verlag, Berlin, 1976, pp. 56–65.

[12] P. Concus, G. H. Golub, and D. P. O'Leary, *A generalised conjugate gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 309–352.

[13] E. J. Craig, *Iteration procedures for simultaneous equations*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, 1954.

[14] J. E. Dennis, Jr. and K. Turner, *Generalized conjugate directions*, Linear Algebra Appl., 88/89 (1987), pp. 187–209.

[15] T. Dupont, R. P. Kendall, and K. K. Rachford Jr., *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM J. Numer. Anal., 5 (1968), pp. 559–573.

[16] S. C. Eisenstat, *Efficient implementation of a class of conjugate methods*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 1–4.

[17] S. C. Eisenstat, H. C. Elman, and M. H. Schultz, *Variational iterative methods for nonsymmetric systems of linear equations*, SIAM J. Numer. Anal., 5 (1983), pp. 345–357.

[18] H. C. Elman, *Iterative methods for large, sparse, nonsymmetric systems of linear equations*, Ph.D. thesis, Computer Science Department, Yale University, 1982.

[19] D. K. Fadeev and V. N. Fadeeva, *Computational Methods of Linear Algebra*, W. H. Freeman, London, 1963.

[20] R. Fletcher, *Conjugate gradient methods for indefinite systems*, in Numerical Analysis Dundee 1975, G. A. Watson, ed., Springer-Verlag, New York, 1976, pp. 73–89.

[21] P. E. Gill and W. Murray, *Newton-type methods for unconstrained and linearly constrained optimization*, Math. Programming, 28 (1974), pp. 311–350.

[22] I. Gustafsson, *A class of first order factorizations*, BIT, 18 (1978), pp. 142–156.

[23] M. R. Hestenes, *The conjugate gradient method for solving linear equations*, in Proc. Sympos. Appl. Math., 6, Numer. Anal., AMS, New York, 1956, pp. 83–102.

[24] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–435.

[25] K. C. Jea, *Generalised conjugate gradient acceleration of iterative methods*, Ph.D. thesis, Mathematics Department, University of Texas, Austin, TX, 1982.

[26] M. P. Kamat, *Nonlinear transient analysis by energy minimization–theoretical basis for the* ACTION *computer code*, NASA Report CR-3287, National Aeronautics and Space Administration, 1980.

[27] M. P. Kamat, L. T. Watson, and J. L. Junkins, *A robust and efficient hybrid method for finding multiple equilibrium solutions*, in Proc. Third Internat. Symposium on Numerical Methods in Engineering, Paris, France, 1983, pp. 799–808.

[28] H. H. Kwok, M. P. Kamat, and L. T. Watson, *Location of stable and unstable equilibrium configurations using a model trust region quasi-Newton method and tunnelling*, Comput. & Structures, 21 (1985), pp. 909–916.

[29] C. Lanczos, *Solution of systems of linear equations by minimized-iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[30] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[31] ———, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as occur in practical problems*, Tech. Report 550, Koninklijke/Shell Exploratie on Produktie Laboratorium, 1980.

[32] D. P. O'Leary, *Hybrid conjugate gradient algorithms*, Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA, 1976.

[33] ———, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[34] J. M. Ortega, *Efficient implementations of certain iterative methods*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 882–891.

[35] W. C. Rheinboldt, *Numerical analysis of continuation methods for nonlinear structural problems*, Comput. & Structures, 13 (1981), pp. 103–113.

[36] ———, *Numerical Analysis of Parametrized Nonlinear Equations*, Wiley-Interscience, New York, 1986.

[37] W. C. Rheinboldt and J. V. Burkardt, *Algorithm* 596: *A program for a locally parameterized continuation process*, ACM Trans. Math. Software, 9 (1983), pp. 236–241.

[38] J. K. Reid, *On the method of conjugate gradients for the solution of sparse systems of linear equations*, in Large Sparse Sets of Linear Equations, J. K. Reid, ed., Academic Press, New York, 1971, pp. 231–254.

[39] Y. Saad, *Krylov subspace methods for solving large unsymmetric linear systems*, Math. Comp., 37 (1981), pp. 105–126.

[40] ———, *Practical use of some Krylov subspace methods for solving indefinite and unsymmetric linear systems*, Tech. Report 214, Department of Computer Science, Yale University, New Haven, CT, 1982.

[41] Y. Saad and M. H. Schultz, *Conjugate gradient-like algorithm for solving nonsymmetric linear systems*, Math. Comp., 44/170 (1985), pp. 417–424.

[42] ———, GMRES: *A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[43] R. S. Varga, *Matrix iterative analysis*, Prentice-Hall, New York, 1962.

[44] P. K. W. VINSOME, *Orthomin, an iterative method for solving sparse sets of simultaneous linear equations*, in Proc. Fourth Symposium on Reservoir Simulation, Society of Petroleum Engineers of the American Institute of Mechanical Engineers, 1976, pp. 149–159.

[45] H. F. WALKER, *Implementations of the* GMRES *method*, Comput. Phys. Comm., 53 (1989), pp. 311–320

[46] ———, *Implementation of the* GMRES *method using Householder transformations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 152–163.

[47] L. T. WATSON, *A globally convergent algorithm for computing fixed points of C^2 maps*, Appl. Math. Comput., 5 (1979), pp. 297–311.

[48] ———, *An algorithm that is globally convergent with probability one for a class of nonlinear two-point boundary value problems*, SIAM J. Numer. Anal., 16 (1979), pp. 394–401.

[49] ———, *Solving finite difference approximations to nonlinear two-point boundary value problems by a homotopy method*, SIAM J. Sci. Statist. Comput., 1 (1980), pp. 467–480.

[50] ———, *Numerical linear algebra aspects of globally convergent homotopy methods*, SIAM Rev., 28 (1986), pp. 529–545.

[51] ———, *Globally convergent homotopy methods: A tutorial*, Tech. Report TR–87–13, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI, 1985.

[52] L. T. WATSON, S. C. BILLUPS, AND A. P. MORGAN, HOMPACK: *A suite of codes for globally convergent homotopy algorithms*, ACM Trans. Math. Software, 13 (1987), pp. 281–310.

[53] L. T. WATSON AND D. FENNER, *Chow–Yorke algorithm for fixed points or zeros of C^2 maps*, ACM Trans. Math. Software, 6 (1980), pp. 252–260.

[54] L. T. WATSON AND M. R. SCOTT, *Solving spline-collocation approximations to nonlinear two-point boundary-value problems by a homotopy method*, Appl. Math. Comput., 24 (1987), pp. 333–357.

[55] L. T. WATSON AND L. R. SCOTT, *Solving Galerkin approximations to nonlinear two-point boundary value problems by a globally convergent homotopy method*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 768–789.

[56] O. WIDLUND, *A Lanczos method of a class of non-symmetric systems of linear equations*, SIAM J. Numer. Anal., 15 (1978), pp. 801–812.

[57] D. M. YOUNG, *Iterative solution of large linear systems*, Academic Press, New York, 1971.

[58] D. M. YOUNG AND K. C. JEA, *Generalised conjugate gradient acceleration of iterative methods. Part 2: the nonsymmetrizable case*, Tech. Report CNA-163, Center for Numerical Analysis, University of Texas, Austin, TX, 1981.

[59] ———, *Generalised conjugate gradient acceleration of nonsymmetrizable iterative methods*, Linear Algebra Appl., 34 (1980), pp. 159–194.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 252–267, May 1991

006

A LOGARITHMIC BARRIER FUNCTION ALGORITHM FOR QUADRATICALLY CONSTRAINED CONVEX QUADRATIC PROGRAMMING*

DONALD GOLDFARB†, SHUCHENG LIU†, AND SIYUN WANG†

Abstract. An interior point method for quadratically constrained convex quadratic programming is presented that is based on a logarithmic barrier function approach and terminates at a required accuracy of an approximate solution in polynomial time. This approach generates a sequence of unconstrained optimization problems, each of which is approximately solved by taking a single step in a Newton direction.

Key words. logarithmic barrier function, interior point method, Newton's method, quadratically constrained convex quadratic programming

AMS(MOS) subject classifications. primary 65K05; secondary 90C30

1. Introduction.

Consider a quadratically constrained convex quadratic programming problem of the form:

$$\text{(QCP)} \qquad \begin{aligned} &\text{minimize} && q(x) \\ &\text{subject to} && q_i(x) = \tfrac{1}{2}x^T G_i x + c_i^T x + b_i \geq 0, \qquad i = 1, \cdots, m, \end{aligned}$$

where x and $c_i, i = 1, \cdots, m \in \mathbf{R}^n$, G_i, $i = 1, \cdots, m$, are symmetric negative semidefinite matrices and $q(x)$ is a convex quadratic function. Without loss of generality, we assume that $q(x) = c^T x$, where $c \in \mathbf{R}^n$, since otherwise, we can introduce an additional variable z and an additional constraint $z \geq q(x)$ and minimize over z. We shall also assume that the interior of the feasible region $S = \{x \in \mathbf{R}^n \mid q_i(x) \geq 0, \quad i = 1, \cdots, m\}$ of (QCP), which we denote as int S, is nonempty and bounded (hence $m \geq 1$). In particular, we shall assume that S is contained within some large sphere of radius R centered at the origin.

Since the publication by Karmarkar [8] of a practical polynomial-time interior point algorithm for linear programming, there have been several papers applying Karmarkar's algorithm and related interior point algorithms to more general classes of problems. For example, for solving convex quadratic programming, Ye and Tse [17] proposed an extension of Karmarkar's projective algorithm; Ben Daya and Shetty [2], Goldfarb and Liu [5], Monteiro and Adler [13], and Ye [18] extended the logarithmic barrier function method of Kojima, Mizuno, and Yoshise [9] and Monteiro and Adler [12]; and Mehrotra and Sun [10] extended Renegar's [14] method of analytic centers.

Recently, Mehrotra and Sun [11] and Jarre [7] proposed two interior point algorithms for solving (QCP) that terminate in a number of iterations which is a polynomial function of m and the logarithm of the accuracy required of an approximate solution. These algorithms are based upon applying Renegar's [14] and Sonnevend's [15] analytic center approach to (QCP). Prior to the development of these two path-following interior point methods, several other algorithms, which do not have

*Received by the editors September 18, 1989, accepted for publication (in revised form) August 24, 1990. This research was supported in part by National Science Foundation grants DMS 85-12277 and CDR 84-21402 and Office of Naval Research contract N-00014-87-K0214.

†Department of Industrial Engineering and Operations Research, Columbia University, New York, New York 10027.

"polynomial-time" worst-case bounds, were proposed for solving problem (QCP) (e.g., see Fang and Rajasekera [3] for references). In this paper, we develop a path-following algorithm for (QCP) using a logarithmic barrier function approach. Specifically, our method solves a sequence of unconstrained optimization problems:

$$(P_{\epsilon^k}) \qquad \min_{x \in \text{int} S} f(x; \epsilon^k) = c^T x - \epsilon^k \sum_{i=1}^{m} \ln(q_i(x)),$$

where the positive barrier parameter ϵ^k satisfies $\epsilon^k \to 0$ as $k \to \infty$. Note that $f(x; \epsilon^k)$ is a function of x which is parameterized by ϵ^k. This logarithmic barrier function approach has previously been used to develop polynomial-time algorithms for linear programming and convex quadratic programming (e.g., see [1], [2], [5], [6], [16]).

From the theory of barrier function methods (e.g., see Fiacco and McCormick [4]), it is well known that under our assumptions on S, P_{ϵ^k} has a unique optimal solution $x_\circ^k$ and that $x_\circ^k$ converges to an optimal solution of the original (QCP) as $\epsilon^k \to 0$. In general, however, the minimizer $x_\circ^k$ cannot be found in "polynomial time." Hence, to develop an algorithm which can be solved up to a required accuracy in polynomial time for (QCP), for each $\epsilon^k > 0$ we will find the point x^k, which is the minimizer of a quadratic approximation to P_{ϵ^k} subject to an ellipsoidal steplength constraint, instead of obtaining $x_\circ^k$. By decreasing ϵ^k slowly and using a "small" steplength ellipsoid, we generate points x^k which are "close to" $x_\circ^k$ in the sense that

$$0 \leq f(x^k; \epsilon^k) - f(x_\circ^k; \epsilon^k) \leq \alpha \epsilon^k \tag{1.1}$$

for some constant $\alpha > 0$, and all k. From (1.1), since $f(x; \epsilon) - f(x_\circ; \epsilon)$ is a strictly convex function on int S (see Lemma 2.1 below) and $\epsilon^k \to 0$ as $k \to \infty$, it follows that $\lim_{k\to\infty} x^k = \lim_{k\to\infty} x_\circ^k = x^*$, where x^* is an optimal solution of (QCP).

This paper is organized as follows. The next section deals with some preliminaries and presents the basic algorithm. In §3, we analyze the convergence of our algorithm by showing that (1.1) is true for all k under some initial assumptions. The last section shows how to find a starting point x^0 and an initial penalty parameter ϵ^0 so that these initial assumptions are satisfied. It is then shown that the algorithm will obtain a feasible solution whose objective value is within $\bar{\epsilon}$ of the optimal value in $O(\sqrt{m} \ln(\epsilon^0/\bar{\epsilon}))$ iterations. The development of our algorithm and our proof of its convergence were greatly influenced by the algorithms and proofs in [1], [2], [6], [7], [11].

2. The algorithm. We define the normalized barrier function for all $x \in$ int S as

$$F(x; \epsilon) = f(x; \epsilon) - f(x_\circ; \epsilon),$$

where $f(x; \epsilon) = c^T x - \epsilon \sum_{i=1}^m \ln(q_i(x))$ and $x_\circ = \text{argmin}_{x \in \text{int} S} f(x; \epsilon)$. Substituting, we obtain

$$F(x; \epsilon) = c^T(x - x_\circ) - \epsilon \sum_{i=1}^{m} \ln \frac{q_i(x)}{q_i(x_\circ)}.$$

Therefore,

$$g \equiv \nabla F(x; \epsilon) = c - \epsilon \sum_{i=1}^{m} \frac{\nabla q_i(x)}{q_i(x)},$$

and

$$H \equiv \nabla^2 F(x; \epsilon) = \epsilon \sum_{i=1}^{m} \left(\frac{\nabla q_i(x) \nabla q_i(x)^T}{(q_i(x))^2} - \frac{G_i}{q_i(x)} \right).$$

Furthermore, we have Lemma 2.1

LEMMA 2.1. *Let $F(x;\epsilon)$ be the function defined above and* int S *be nonempty and bounded. Then*

(1°) $F(x;\epsilon)$ *is strictly convex on* int S, *and*

(2°) $c = \epsilon \sum_{i=1}^{m} \frac{\nabla q_i(x_\circ)}{q_i(x_\circ)}$.

Proof. (1°) Clearly, $\nabla^2 F(x;\epsilon)$ is at least positive-semidefinite for all $x \in$ int S. If, for some $x \in$ int S, $\nabla^2 F(x;\epsilon)$ is not positive-definite, there exists a nonzero vector d such that $d^T \nabla^2 F(x;\epsilon) d = 0$, which implies that $\nabla q_i(x)^T d = 0$ and $d^T G_i d = 0$ for all $i = 1, \cdots, m$. Hence

$$q_i(x + \lambda d) = q_i(x) + \lambda \nabla q_i(x)^T d + \frac{\lambda^2}{2} d^T G_i d = q_i(x) \qquad \forall \lambda \in \mathbf{R},$$

for all $i = 1, \cdots, m$, which means that the line $x + \lambda d$ $(-\infty < \lambda < +\infty)$ lies entirely in int S. This contradicts the boundedness of S.

(2°) Since $F(x;\epsilon)$ is minimized at $x_\circ$,

$$\nabla F(x_\circ;\epsilon) = c - \epsilon \sum_{i=1}^{m} \frac{\nabla q_i(x_\circ)}{q_i(x_\circ)} = 0. \qquad \square$$

We now present our basic algorithm. Values for the parameters α, δ, and σ used in the algorithm are specified in Theorem 3.9 and one way to obtain an appropriate starting point x^0 and an initial barrier parameter ϵ^0 is discussed in §4.

ALGORITHM 2.1

Step 0. Given a point $x^0 \in$ int S, $\epsilon^0 > 0$ and $\alpha > 0$ such that $F(x^0;\epsilon^0) \leq \alpha\epsilon^0$, choose σ, $\delta \in (0,1)$ and $\bar{\epsilon} \leq \epsilon^0/m$, and set $k = 1$.

Step 1. Set $\epsilon^k = (1 - (\sigma/\sqrt{m}))\epsilon^{k-1}$. Determine the Newton direction h_{k-1} by solving

$$H_k h_{k-1} = -g_k,$$

where H_k and g_k are the Hessian and gradient of $F(x;\epsilon^k)$ evaluated at x^{k-1}, respectively.

Step 2. Set

$$x^k = x^{k-1} + \lambda_{k-1} h_{k-1},$$

where the steplength

$$\lambda_{k-1} = \sqrt{\frac{\delta^2 \epsilon^k}{h_{k-1}^T H_k h_{k-1}}} \quad .$$

Step 3. If $\epsilon^k \leq \bar{\epsilon}$, STOP! else set $k := k + 1$, goto Step 1.

We now give several lemmas which will be useful in proving the convergence of Algorithm 2.1 in §3. First let us define

$$E(y,\delta) = \{x \mid (x-y)^T \nabla^2 F(y;\epsilon)(x-y) \leq \delta^2 \epsilon\}, \qquad \epsilon > 0.$$

Note that the ellipsoid $E(y,\delta)$ does not depend upon ϵ since $1/\epsilon\nabla^2 F(y;\epsilon)$ does not.

LEMMA 2.2. *If* $x \in E(y,\delta) \cap S$, *where* $y \in \text{int } S$, *then*

$$\sum_{i=1}^{m} \left(\frac{q_i(x) - q_i(y)}{q_i(y)} \right)^2 \leq \left(\delta + \frac{1}{2}\delta^2 \right)^2, \tag{2.1}$$

$$\sum_{i=1}^{m} \ln \frac{q_i(x)}{q_i(y)} \leq \sum_{i=1}^{m} \frac{|\nabla q_i(y)^T (x-y)|}{q_i(y)} \leq \delta\sqrt{m}\ . \tag{2.2}$$

Furthermore,

$$E(y,\delta) \subseteq \text{int } S \quad \textit{if} \quad 0 < \delta < 0.7\ .$$

Proof. Let $u = (u_1, u_2, \cdots, u_m)^T$, $v = (v_1, \cdots, v_m)^T$, and $w = (w_1, \cdots, w_m)^T$, where

$$u_i = \frac{q_i(x) - q_i(y)}{q_i(y)}, \quad v_i = \frac{\nabla q_i(y)^T (x-y)}{q_i(y)}, \quad w_i = \frac{(x-y)^T G_i (x-y)}{q_i(y)}.$$

Then (2.1) and (2.2) can be stated as

$$\|u\|_2 \leq \delta + \tfrac{1}{2}\delta^2 \quad \text{and} \quad \|v\|_1 \leq \delta\sqrt{m}.$$

Since $x \in E(y,\delta)$, we have from the definition of $E(y,\delta)$ and H that

$$\epsilon \sum_{i=1}^{m} \left(\left(\frac{\nabla q_i(y)^T (x-y)}{q_i(y)} \right)^2 - \frac{(x-y)^T G_i (x-y)}{q_i(y)} \right) \leq \delta^2 \epsilon, \qquad \epsilon > 0,$$

which implies that

$$\|v\|_2 \leq \delta \quad \text{and} \quad \|w\|_1 \leq \delta^2. \tag{2.3}$$

Note that, for $i = 1, \cdots, m$,

$$u_i = \frac{q_i(x) - q_i(y)}{q_i(y)} = \frac{\nabla q_i(y)^T (x-y)}{q_i(y)} + \frac{1}{2} \frac{(x-y)^T G_i (x-y)}{q_i(y)} = v_i + \frac{1}{2} w_i.$$

Therefore, $u = v + \frac{1}{2}w$, and it follows from the triangle inequality and the well-known fact that $\|z\|_2 \leq \|z\|_1 \leq \sqrt{m}\|z\|_2$, for all $z \in \mathbf{R}^m$, that

$$\|u\|_2 \leq \|v\|_2 + \tfrac{1}{2}\|w\|_2 \leq \delta + \tfrac{1}{2}\|w\|_1 \leq \delta + \tfrac{1}{2}\delta^2$$

and

$$\|v\|_1 \leq \sqrt{m}\|v\|_2 \leq \delta\sqrt{m}.$$

To verify the first inequality in (2.2) we note that

$$\ln \frac{q_i(x)}{q_i(y)} = \ln \left(1 + \frac{q_i(x) - q_i(y)}{q_i(y)} \right) \leq \frac{q_i(x) - q_i(y)}{q_i(y)} \leq \frac{\nabla q_i(y)^T (x-y)}{q_i(y)}.$$

Furthermore, since $| u_i | = (| q_i(x) - q_i(y) |)/q_i(y) \leq \|u\|_2 \leq \delta + \frac{1}{2}\delta^2$, for any $x \in E(y, \delta)$, where $0 < \delta < 0.7$, we have, for $i = 1, \cdots, m$,

$$\frac{q_i(x)}{q_i(y)} = 1 + \frac{q_i(x) - q_i(y)}{q_i(y)} \geq 1 - \delta - \frac{1}{2}\delta^2 > 0, \tag{2.4}$$

and hence that $x \in \text{int } S$. Therefore,

$$E(y, \delta) \subseteq \text{int } S \quad \text{if } 0 < \delta < 0.7 . \qquad \square$$

In Step 2 of Algorithm 2.1, the steplength λ_{k-1} is chosen so that the point x^k lies on the boundary of $E(x^{k-1}, \delta)$. Therefore, it immediately follows from Lemma 2.2 that we have Corollary 2.3.

COROLLARY 2.3. *Let $x^{k-1} \in \text{int } S$. If x^k is obtained from x^{k-1} as in Algorithm 2.1 and $0 < \delta < 0.7$, then $x^k \in \text{int } S$.* $\square$

The proof of the following lemma can be found in Gonzaga [6].

LEMMA 2.4. *Assume that $\|u\|_2 < 1$, $u \in \mathbf{R}^m$, and $p(x) = -\sum_{i=1}^m \ln x_i$. Then*

$$p(e + u) = -e^T u + \tfrac{1}{2}\|u\|_2^2 + o(u),$$

where $|o(u)| \leq (\|u\|_2^3/3(1 - \|u\|_2))$ and $e = (1, \cdots, 1)^T \in \mathbf{R}^m$. $\square$

Now, consider the quadratic Taylor series approximation to $F(x; \epsilon)$ at a point $y \in \text{int } S$

$$Q_y(x; \epsilon) = F(y; \epsilon) + \nabla F(y; \epsilon)^T (x - y) + \tfrac{1}{2}(x - y)^T \nabla^2 F(y; \epsilon)(x - y). \tag{2.5}$$

The next lemma gives a bound for the difference between $Q_y(x; \epsilon)$ and $F(x; \epsilon)$.

LEMMA 2.5. *If $x \in E(y, \delta)$, where $y \in \text{int } S$ and $0 < \delta < 0.7$, then*

$$|F(x; \epsilon) - Q_y(x; \epsilon)| \leq \bar{K}(\delta)\epsilon,$$

where

$$\bar{K}(\delta) = \frac{\delta^3}{2} + \frac{\delta^4}{8} + \frac{(\delta + \frac{1}{2}\delta^2)^3}{3(1 - \delta - \frac{1}{2}\delta^2)}. \tag{2.6}$$

Furthermore, $\bar{K}(\delta) \leq \delta^3$, if $\delta \leq \frac{1}{9}$.

Proof. Let u, v, and $w \in \mathbf{R}^m$ be defined as in the proof of Lemma 2.2. Since $q_i(x)/q_i(y) = 1 + (q_i(x) - q_i(y))/q_i(y) = 1 + u_i$ where $\|u\|_2 \leq \delta + \frac{1}{2}\delta^2 < 1$ by Lemma 2.2 and

$$q_i(x) - q_i(y) = \nabla q_i(y)^T (x - y) + \tfrac{1}{2}(x - y)^T G_i (x - y), \tag{2.7}$$

it follows from Lemma 2.4 that

$$\begin{aligned}
F(x;\epsilon) - F(y;\epsilon) &= c^T(x-y) - \epsilon \sum_{i=1}^{m} \ln\left(1 + \frac{q_i(x) - q_i(y)}{q_i(y)}\right) \\
&= c^T(x-y) - \epsilon \sum_{i=1}^{m} \frac{\nabla q_i(y)^T(x-y) + \frac{1}{2}(x-y)^T G_i(x-y)}{q_i(y)} \\
&\quad + \frac{\epsilon}{2} \sum_{i=1}^{m} \left(\frac{q_i(x) - q_i(y)}{q_i(y)}\right)^2 + o(u)\epsilon \\
&= \nabla F(y;\epsilon)^T(x-y) - \frac{\epsilon}{2} \sum_{i=1}^{m} \frac{(x-y)^T G_i(x-y)}{q_i(y)} \\
&\quad + \frac{\epsilon}{2} \sum_{i=1}^{m} \left(\frac{\nabla q_i(y)^T(x-y) + \frac{1}{2}(x-y)^T G_i(x-y)}{q_i(y)}\right)^2 + o(u)\epsilon \\
&= \nabla F(y;\epsilon)^T(x-y) \\
&\quad + \frac{\epsilon}{2} \sum_{i=1}^{m} \left(\left(\frac{\nabla q_i(y)^T(x-y)}{q_i(y)}\right)^2 - \frac{(x-y)^T G_i(x-y)}{q_i(y)}\right) \\
&\quad + \frac{\epsilon}{2} \sum_{i=1}^{m} \frac{\nabla q_i(y)^T(x-y)(x-y)^T G_i(x-y)}{(q_i(y))^2} \\
&\quad + \frac{\epsilon}{8} \sum_{i=1}^{m} \left(\frac{(x-y)^T G_i(x-y)}{q_i(y)}\right)^2 + o(u)\epsilon \\
&= \nabla F(y;\epsilon)^T(x-y) + \frac{1}{2}(x-y)^T \nabla^2 F(y;\epsilon)(x-y) \\
&\quad + \frac{\epsilon}{2} \sum_{i=1}^{m} \frac{\nabla q_i(y)^T(x-y)(x-y)^T G_i(x-y)}{(q_i(y))^2} \\
&\quad + \frac{\epsilon}{8} \sum_{i=1}^{m} \left(\frac{(x-y)^T G_i(x-y)}{q_i(y)}\right)^2 + o(u)\epsilon,
\end{aligned}$$

where

$$|o(u)| \le \frac{\|u\|_2^3}{3(1 - \|u\|_2)} \le \frac{(\delta + \frac{1}{2}\delta^2)^3}{3(1 - \delta - \frac{1}{2}\delta^2)}.$$

Consequently, by (2.3),

$$\begin{aligned}
|F(x;\epsilon) - Q_y(x;\epsilon)| &\le \frac{\epsilon}{2}\left|\sum_{i=1}^{m} v_i w_i\right| + \frac{\epsilon}{8}\sum_{i=1}^{m} w_i^2 + |o(u)|\epsilon \\
&\le \frac{\epsilon}{2}\|v\|_2\|w\|_2 + \frac{\epsilon}{8}\|w\|_2^2 + |o(u)|\epsilon \\
&\le \frac{\epsilon}{2}\|v\|_2\|w\|_1 + \frac{\epsilon}{8}\|w\|_1^2 + |o(u)|\epsilon \\
&\le \frac{\epsilon}{2}\delta^3 + \frac{\epsilon}{8}\delta^4 + \frac{(\delta + \frac{1}{2}\delta^2)^3}{3(1 - \delta - \frac{1}{2}\delta^2)}\epsilon = \bar{K}(\delta)\epsilon. \qquad \square
\end{aligned}$$

The following lemma shows that the closeness of x and x_o^k can be measured by the value of the normalized barrier function.

LEMMA 2.6. *Let* $0 < \delta < \frac{1}{3}$ *and* $F(x;\epsilon^k) \leq ((\delta^2/2) - \bar{K}(\delta))\epsilon^k$, *where* $\bar{K}(\delta)$ *is defined by* (2.6). *Then*

$$x \in E(x_\circ^k, \delta).$$

Proof. First note that $(\delta^2/2) > \bar{K}(\delta)$ for $0 < \delta < \frac{1}{3}$. Since $F(x;\epsilon^k)$ is strictly convex on int S and achieves its minimum value at $x_\circ^k$, the center of $E(x_\circ^k, \delta)$, it is sufficient to show that

$$F(x;\epsilon^k) \geq \left(\frac{\delta^2}{2} - \bar{K}(\delta)\right)\epsilon^k$$

for all points x on the boundary $\partial E(x_\circ^k, \delta)$ of $E(x_\circ^k, \delta)$. But for any such x, by Lemma 2.5, we have

$$\begin{aligned} F(x;\epsilon^k) &\geq F(x_\circ^k;\epsilon^k) + \nabla F(x_\circ^k;\epsilon^k)^T(x - x_\circ^k) \\ &\quad + \frac{1}{2}(x - x_\circ^k)^T\nabla^2 F(x_\circ^k;\epsilon^k)(x - x_\circ^k) - \bar{K}(\delta)\epsilon^k \\ &= \frac{\delta^2}{2}\epsilon^k - \bar{K}(\delta)\epsilon^k, \end{aligned}$$

where the last equality follows from the facts that $F(x_\circ^k;\epsilon^k) = 0$ and $\nabla F(x_\circ^k;\epsilon^k) = 0$. □

3. Convergence analysis. As indicated in §1, if $F(x^k;\epsilon^k) \leq \alpha\epsilon^k$ for all k, then the sequence $\{x^k\}$ converges to an optimal solution of (QCP). We now show that there are positive values α, δ, and σ such that given a feasible point x^{k-1} and penalty parameter ϵ^{k-1}, satisfying $F(x^{k-1};\epsilon^{k-1}) \leq \alpha\epsilon^{k-1}$, Algorithm 2.1 generates a feasible point x^k and penalty parameter ϵ^k satisfying $F(x^k;\epsilon^k) \leq \alpha\epsilon^k$. First we show in Lemma 3.1 below that $F(x^k;\epsilon^k)$ can be written as the sum of three terms. Then we provide bounds for the first two of these terms in the following two sections and combine them and bound the third term to give the desired bound on $F(x^k;\epsilon^k)$ in §3.3.

LEMMA 3.1. *Let* $\epsilon^k = (1 - (\sigma/\sqrt{m}))\epsilon^{k-1}, 0 < \sigma < 1$. *Then*

$$F(x^k;\epsilon^k) = F(x^k;\epsilon^{k-1}) + F(x_\circ^{k-1};\epsilon^k) + \frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^m \ln\frac{q_i(x^k)}{q_i(x_\circ^{k-1})}. \tag{3.1}$$

Proof. The identity (3.1) is easily verified from the definition of $F(x;\epsilon)$. □

3.1. Bounding $F(x^k;\epsilon^{k-1})$. The following lemma and corollary are preparatory for bounding $F(x^k;\epsilon^{k-1})$.

LEMMA 3.2. *Let* $x \in E(x_\circ^{k-1}, \delta_1)$, *where* $0 < \delta_1 < \frac{1}{2}$. *Then*

$$\nabla F(x;\epsilon^{k-1})^T(x - x_\circ^{k-1}) \geq \delta_2(x - x_\circ^{k-1})^T\nabla^2 F(x;\epsilon^{k-1})(x - x_\circ^{k-1}), \tag{3.2}$$

where $\delta_2 = 1 - \frac{3}{2}\delta_1 - \delta_1^2$.

Proof. From the definition of $\nabla F(x;\epsilon^{k-1})$ and Lemma 2.1, we have

$$\nabla F(x;\epsilon^{k-1}) = c - \epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x)}{q_i(x)} = \epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x_\circ^{k-1})}{q_i(x_\circ^{k-1})} - \epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x)}{q_i(x)}.$$

Letting $x_0 = x_\circ^{k-1}$, $\triangle x = x - x_0$, and $r_i = q_i(x)/q_i(x_0)$, we can write the above gradient as

$$\nabla F(x;\epsilon^{k-1}) = \epsilon^{k-1}\sum_{i=1}^m \rho_i, \tag{3.3}$$

where

$$\rho_i = \frac{\nabla q_i(x_0)}{q_i(x_0)} - \frac{\nabla q_i(x)}{q_i(x)} = \frac{r_i \nabla q_i(x_0) - \nabla q_i(x)}{q_i(x)} = \frac{(r_i - 1)\nabla q_i(x) - r_i G_i \triangle x}{q_i(x)},$$

since

$$\nabla q_i(x) = \nabla q_i(x_0) + G_i \triangle x. \tag{3.4}$$

From (2.7), with x replaced by x_0 and y by x, we have that

$$r_i - 1 = r_i \frac{q_i(x) - q_i(x_0)}{q_i(x)} = r_i \frac{\nabla q_i(x)^T \triangle x - \frac{1}{2} \triangle x^T G_i \triangle x}{q_i(x)}.$$

Hence,

$$\begin{aligned} \rho_i^T \triangle x &= r_i \left(\frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right)^2 - r_i \frac{\triangle x^T G_i \triangle x}{q_i(x)} \left(1 + \frac{1}{2} \frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right) \\ &= r_i \left(\frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right)^2 - \beta_i \frac{\triangle x^T G_i \triangle x}{q_i(x)}, \end{aligned} \tag{3.5}$$

where, using (2.7) with y replaced by x_0 and (3.4),

$$\beta_i = r_i \left(1 + \frac{1}{2} \frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right) = \frac{q_i(x_0) + \frac{3}{2} \nabla q_i(x_0)^T \triangle x + \triangle x^T G_i \triangle x}{q_i(x_0)}.$$

Now, it follows from (2.3) and (2.4) in the proof of Lemma 2.2 that $r_i \geq 1 - \delta_1 - \frac{1}{2}\delta_1^2 \geq \delta_2$ and $\beta_i \geq 1 - \frac{3}{2}\delta_1 - \delta_1^2 = \delta_2$, for $i = 1, \cdots, m$. Therefore, after combining (3.3) and (3.5) we obtain

$$\begin{aligned} \nabla F(x; \epsilon^{k-1})^T \triangle x &= \epsilon^{k-1} \sum_{k=1}^{m} \left(r_i \left(\frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right)^2 - \beta_i \frac{\triangle x^T G_i \triangle x}{q_i(x)} \right) \\ &\geq \epsilon^{k-1} \delta_2 \sum_{k=1}^{m} \left(\left(\frac{\nabla q_i(x)^T \triangle x}{q_i(x)} \right)^2 - \frac{\triangle x^T G_i \triangle x}{q_i(x)} \right) \\ &= \delta_2 \triangle x^T \nabla^2 F(x; \epsilon^{k-1}) \triangle x. \end{aligned}$$

□

From the above lemma, we immediately have the following corollary.

COROLLARY 3.3. *Let* $x \in E(x_\circ^{k-1}, \delta_1)$, *where* $0 < \delta_1 < \frac{1}{2}$. *Then*

$$\nabla F(x; \epsilon^{k-1})^T (x - x_\circ^{k-1}) \geq \sqrt{\delta_2 F(x; \epsilon^{k-1})(x - x_\circ^{k-1})^T \nabla^2 F(x; \epsilon^{k-1})(x - x_\circ^{k-1})},$$

where $\delta_2 = (1 - \frac{3}{2}\delta_1 - \delta_1^2) > 0$.

Proof. Since $F(x_\circ^{k-1}; \epsilon^{k-1}) = 0$ and $F(\cdot; \epsilon^{k-1})$ is convex, we have

$$0 \leq F(x; \epsilon^{k-1}) = F(x; \epsilon^{k-1}) - F(x_\circ^{k-1}; \epsilon^{k-1}) \leq \nabla F(x; \epsilon^{k-1})^T (x - x_\circ^{k-1}).$$

Combining this inequality with (3.2) in Lemma 3.2 , we get the required result. □

From the definitions of $\nabla F(x;\epsilon)$ and $\nabla^2 F(x;\epsilon)$, and the relation

$$\epsilon^k = (1 - (\sigma/\sqrt{m}))\epsilon^{k-1},$$

we have for all $x \in \text{int } S$,

$$\nabla^2 F(x;\epsilon^k) = \frac{\epsilon^k}{\epsilon^{k-1}} \nabla^2 F(x;\epsilon^{k-1}), \tag{3.6}$$

$$\nabla F(x;\epsilon^k) = \nabla F(x;\epsilon^{k-1}) + \frac{\sigma}{\sqrt{m}}\epsilon^{k-1} \sum_{i=1}^{m} \frac{\nabla q_i(x)}{q_i(x)}. \tag{3.7}$$

Now, we are ready to give a bound on $F(x^k;\epsilon^{k-1})$.

THEOREM 3.4. *Let $0 < \delta < 0.7$, $0 < \delta_1 < \frac{1}{3}$ and $\alpha \le (\delta_1^2/2) - \bar{K}(\delta_1)$, where $\bar{K}(\cdot)$ is defined by (2.6). Assume that $F(x^{k-1};\epsilon^{k-1}) \le \alpha\epsilon^{k-1}$, $x^{k-1} \in \text{int } S$ and x^k is obtained from x^{k-1}, as in Algorithm 2.1. Then*

$$F(x^k;\epsilon^{k-1}) \le F(x^{k-1};\epsilon^{k-1}) - \sqrt{\epsilon^{k-1}}\delta\sqrt{\delta_2 F(x^{k-1};\epsilon^{k-1})} + \left(\frac{\delta^2}{2} + 2\sigma\delta + \bar{K}(\delta)\right)\epsilon^{k-1},$$

where $\delta_2 = 1 - \frac{3}{2}\delta_1 - \delta_1^2$.

Proof. Since $F(x^{k-1};\epsilon^{k-1}) \le \alpha\epsilon^{k-1} \le \left((\delta_1^2/2) - \bar{K}(\delta_1)\right)\epsilon^{k-1}$, by Lemma 2.6, we have that $x^{k-1} \in E(x_\circ^{k-1}, \delta_1)$, and hence, from Corollary 3.3, we obtain

$$\begin{aligned} &\nabla F(x^{k-1};\epsilon^{k-1})^T(x^{k-1} - x_\circ^{k-1}) \\ &\ge \sqrt{\delta_2 F(x^{k-1};\epsilon^{k-1})(x^{k-1} - x_\circ^{k-1})^T \nabla^2 F(x^{k-1};\epsilon^{k-1})(x^{k-1} - x_\circ^{k-1})} \end{aligned} \tag{3.8}$$

Because x^k is the minimizer of $\nabla F(x^{k-1};\epsilon^{k-1})^T y$ over $E(x^{k-1},\delta)$ and $0 < \delta < 0.7$, we have, from Lemma 2.5 and (3.6), that

$$\begin{aligned} F(x^k;\epsilon^{k-1}) &\le F(x^{k-1};\epsilon^{k-1}) + \nabla F(x^{k-1};\epsilon^{k-1})^T(x^k - x^{k-1}) \\ &\quad + \tfrac{1}{2}(x^k - x^{k-1})^T \nabla^2 F(x^{k-1};\epsilon^{k-1})(x^k - x^{k-1}) + \bar{K}(\delta)\epsilon^{k-1}. \\ &\le F(x^{k-1};\epsilon^{k-1}) + \nabla F(x^{k-1};\epsilon^{k-1})^T(x^k - x^{k-1}) + \tfrac{1}{2}\delta^2\epsilon^{k-1} + \bar{K}(\delta)\epsilon^{k-1}. \end{aligned} \tag{3.9}$$

Moreover, from (3.7) and Lemma 2.2, it follows that

$$\begin{aligned} &\nabla F(x^{k-1};\epsilon^{k-1})^T(x^k - x^{k-1}) \\ &= \nabla F(x^{k-1};\epsilon^k)^T(x^k - x^{k-1}) - \frac{\sigma}{\sqrt{m}}\epsilon^{k-1} \sum_{i=1}^{m} \frac{\nabla q_i(x^{k-1})^T(x^k - x^{k-1})}{q_i(x^{k-1})} \\ &\le \nabla F(x^{k-1};\epsilon^k)^T(x^k - x^{k-1}) + \sigma\delta\epsilon^{k-1}. \end{aligned} \tag{3.10}$$

The next step is to estimate $\nabla F(x^{k-1};\epsilon^k)^T(x^k - x^{k-1})$. Suppose now that $x_\circ^{k-1} \ne x^{k-1}$ since, otherwise, from (3.9), $F(x^k;\epsilon^{k-1}) \le \frac{1}{2}\delta^2\epsilon^{k-1} + \bar{K}(\delta)\epsilon^{k-1}$, which implies that the theorem holds. Let $\bar{x}$ be the point where the semi-infinite ray $\{x \mid x = x^{k-1} + \lambda(x_\circ^{k-1} - x^{k-1}), \lambda \ge 0\}$ intersects the boundary of the ellipsoid $E(x^{k-1},\delta)$. Since $\bar{x} \in E(x^{k-1},\delta)$, by using Lemma 2.2 once more, we have

$$\sum_{i=1}^{m} \frac{|\nabla q_i(x^{k-1})^T(\bar{x} - x^{k-1})|}{q_i(x^{k-1})} \le \delta\sqrt{m}.$$

Due to the fact that $\nabla F(x^{k-1};\epsilon^k)^T x^k \leq \nabla F(x^{k-1};\epsilon^k)^T \bar{x}$, it follows from (3.7) and the above bound that

$$\begin{aligned}\nabla F(x^{k-1};\epsilon^k)^T(x^k - x^{k-1}) &\leq \nabla F(x^{k-1};\epsilon^k)^T(\bar{x} - x^{k-1}) \\ &= \nabla F(x^{k-1};\epsilon^{k-1})^T(\bar{x} - x^{k-1}) \\ &\quad + \frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^{m}\frac{\nabla q_i(x^{k-1})^T(\bar{x} - x^{k-1})}{q_i(x^{k-1})} \\ &\leq \nabla F(x^{k-1};\epsilon^{k-1})^T(\bar{x} - x^{k-1}) + \sigma\delta\epsilon^{k-1}.\end{aligned}$$

Since $\bar{x} - x^{k-1} = \bar{\lambda}(x_\circ^{k-1} - x^{k-1})$, where

$$\bar{\lambda} = \frac{\sqrt{\epsilon^k}\delta}{((x_\circ^{k-1} - x^{k-1})^T\nabla^2 F(x^{k-1};\epsilon^k))(x_\circ^{k-1} - x^{k-1})^{1/2}},$$

it then follows from (3.6) and (3.8) that

$$\nabla F(x^{k-1};\epsilon^k)^T(x^k - x^{k-1}) \leq -\sqrt{\epsilon^{k-1}}\delta\sqrt{\delta_2 F(x^{k-1};\epsilon^{k-1})} + \sigma\delta\epsilon^{k-1}. \tag{3.11}$$

Combining (3.9), (3.10), and (3.11) yields the required result. □

3.2. Bounding $F(x_\circ^{k-1};\epsilon^k)$. We first need to prove Lemma 3.5 below which is a slight generalization and extension of Lemma 2.5 in Gonzaga [6]. Let us define $\bar{F}(x;\epsilon) = F(x;\epsilon)/\epsilon$ and $\bar{Q}_y(x;\epsilon) = Q_y(x;\epsilon)/\epsilon$, where $Q_y(x;\epsilon)$ is the quadratic Taylor series approximation (2.5) to $F(x;\epsilon)$ at the point $y \in \text{int } S$. Furthermore, let $\|x\|_{H_y} = (x^T H_y x)^{1/2}$, where $H_y \equiv \nabla^2\bar{F}(y;\epsilon)$, and let $x_N \equiv y + h_N \equiv \text{argmin } \bar{Q}_y(x;\epsilon)$ and $\bar{x} \equiv y + \bar{h} \equiv \text{argmin } \bar{F}(x;\epsilon)$ for $x \in \text{int } S$. We have the following result.

LEMMA 3.5. *If* $0 < \|h_N\|_{H_y} = \delta_1 \leq \delta < (1/18)$, *then*

$$\|\bar{x} - x_N\|_{H_y} = \|\bar{h} - h_N\|_{H_y} \leq \bar{\alpha}\|h_N\|_{H_y}$$

and

$$\|\bar{x} - y\|_{H_y} = \|\bar{h}\|_{H_y} \leq (1 + \bar{\alpha})\|h_N\|_{H_y},$$

where $\bar{\alpha} \equiv \sqrt{18\|h_N\|_{H_y}} < 1$.

Proof. First note that $\bar{x} - y = \bar{x} - x_N + h_N$; hence the second result of the lemma follows from the first result by the triangle inequality. Also, since $x_N \in E(y,\delta_1) \subseteq E(y,\delta)$, it follows from Lemma 2.5 that

$$\bar{F}(x_N;\epsilon) \leq \bar{Q}_y(x_N;\epsilon) + \bar{K}(\delta_1), \tag{3.12}$$

where $\bar{K}(\delta_1) \leq \delta_1^3$. Since $\bar{Q}_y(x;\epsilon)$ is minimized at x_N, $\nabla\bar{Q}_y(x_N;\epsilon) = 0$, and

$$\bar{Q}_y(z;\epsilon) = \bar{Q}_y(x_N;\epsilon) + \tfrac{1}{2}\|z - x_N\|_{H_y}^2, \tag{3.13}$$

for any point z. Now, by contradiction, suppose that $\|\bar{x} - x_N\|_{H_y} = \|\bar{h} - h_N\|_{H_y} > \bar{\alpha}\|h_N\|_{H_y}$. Then we can choose a point z on the line segment joining x_N and $\bar{x}$ such that

$$\|z - x_N\|_{H_y} = \bar{\alpha}\|h_N\|_{H_y}. \tag{3.14}$$

Let $d = z - x_N$. Then $\|h_N + d\|_{H_y} \leq (1 + \bar{\alpha})\|h_N\|_{H_y} = \delta_\circ < (1/9)$. Hence, from Lemma 2.5, we have that

$$\bar{Q}_y(z;\epsilon) - \bar{K}(\delta_\circ) \leq \bar{F}(z;\epsilon), \tag{3.15}$$

where $\bar{K}(\delta_\circ) \leq \delta_\circ^3$. Moreover, $\bar{F}(x_N;\epsilon) \geq \bar{F}(z;\epsilon)$, since $\bar{F}(x;\epsilon)$ is minimized at $\bar{x}$ and $\bar{F}(x;\epsilon)$ is strictly convex. From this and (3.12)–(3.15) we then obtain

$$\begin{aligned}\tfrac{1}{2}\bar{\alpha}^2\|h_N\|_{H_y}^2 + \bar{Q}_y(x_N;\epsilon) = \bar{Q}_y(z;\epsilon) &\leq \bar{F}(z;\epsilon) + \bar{K}(\delta_\circ) \\ &\leq \bar{F}(x_N;\epsilon) + \bar{K}(\delta_\circ) \leq \bar{Q}_y(x_N;\epsilon) + \bar{K}(\delta_1) + \bar{K}(\delta_\circ),\end{aligned}$$

which, since $\bar{\alpha} < 1$, implies that

$$\tfrac{1}{2}\bar{\alpha}^2\|h_N\|_{H_y}^2 \leq \bar{K}(\delta_1) + \bar{K}(\delta_\circ) \leq \delta_1^3 + \delta_\circ^3 \leq \|h_N\|_{H_y}^3[1 + (1 + \bar{\alpha})^3] < 9\|h_N\|_{H_y}^3.$$

But $\frac{1}{2}\bar{\alpha}^2\|h_N\|_{H_y}^2 = 9\|h_N\|_{H_y}^3$; hence our assumption that $\|\bar{x} - x_N\|_{H_y} > \bar{\alpha}\|h_N\|_{H_y}$ is false. □

LEMMA 3.6. *If* $0 < \sigma < 1$, *then*

$$\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}} \leq \frac{\sigma}{1-\sigma},$$

where $h_N^{k-1} = x_N^{k-1} - x_\circ^{k-1}$ *and* $x_N^{k-1} \equiv \operatorname{argmin} \bar{Q}_{x_\circ^{k-1}}(x;\epsilon^k) \equiv \operatorname{argmin} Q_{x_\circ^{k-1}}(x;\epsilon^k)$.

Proof. Since h_N^{k-1} satisfies $\nabla^2 F(x_\circ^{k-1};\epsilon^k)h_N^{k-1} = -\nabla F(x_\circ^{k-1};\epsilon^k)$,

$$\begin{aligned}\epsilon^k\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}}^2 &= \left(h_N^{k-1}\right)^T \nabla^2 F(x_\circ^{k-1};\epsilon^k)h_N^{k-1} = -\left(h_N^{k-1}\right)^T \nabla F(x_\circ^{k-1};\epsilon^k) \\ &= -\frac{\sigma\epsilon^{k-1}}{\sqrt{m}} \sum_{i=1}^{m} \frac{\left(h_N^{k-1}\right)^T \nabla q_i(x_\circ^{k-1})}{q_i(x_\circ^{k-1})},\end{aligned}$$

where the last equality follows from (3.7) and the fact that $\nabla F(x_\circ^{k-1};\epsilon^{k-1}) = 0$.

Let $y_i = \left(\left(h_N^{k-1}\right)^T \nabla q_i(x_\circ^{k-1})\right)/q_i(x_\circ^{k-1})$, for $i = 1,\cdots,m$. Then since G_i, $i = 1,\cdots,m$, are negative semidefinite

$$-e^T y \leq \|y\|_1 \leq \sqrt{m}\|y\|_2 \leq \sqrt{m}\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}},$$

and it follows that

$$\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}}^2 = \frac{\sigma/\sqrt{m}}{1-(\sigma/\sqrt{m})}(-e^T y) \leq \frac{\sigma/\sqrt{m}}{1-(\sigma/\sqrt{m})}\sqrt{m}\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}},$$

and hence that

$$\|h_N^{k-1}\|_{H_{x_\circ^{k-1}}} \leq \frac{\sigma}{1-\sigma}.$$

□

COROLLARY 3.7. *If* $0 < \sigma < (1/19)$, *then* $x_\circ^k \in E(x_\circ^{k-1}, \delta_3)$, *where*

$$\delta_3 = \left(1 + \sqrt{\frac{18\sigma}{1-\sigma}}\right)\left(\frac{\sigma}{1-\sigma}\right).$$

Proof. Since $\sigma/(1-\sigma) < (1/18)$ for $0 < \sigma < (1/19)$, and $h_N^{k-1} \neq 0$, it follows from Lemma 3.6 and Lemma 3.5, with $\bar{x}$, y, and h_N replaced by $x_\circ^k$, $x_\circ^{k-1}$, and h_N^{k-1}, respectively, that $\|x_\circ^k - x_\circ^{k-1}\|_{H_{x_\circ^{k-1}}} \leq \delta_3$. □

Now, we are ready to prove the main theorem of this section.

THEOREM 3.8. *If* $0 < \sigma < (1/19)$, *then* $F(x_\circ^{k-1}; \epsilon^k) \leq \sigma\delta_3\epsilon^{k-1}$, *where*

$$\delta_3 = \left(1 + \sqrt{\frac{18\sigma}{1-\sigma}}\right)\left(\frac{\sigma}{1-\sigma}\right).$$

Proof. From Lemma 2.1,

$$c^T(x_\circ^{k-1} - x_\circ^k) = \epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x_\circ^{k-1})^T(x_\circ^{k-1} - x_\circ^k)}{q_i(x_\circ^{k-1})}.$$

So, from the definition of $F(x_\circ^{k-1}; \epsilon^k)$ and the fact that $\ln(1+x) \leq x$ for $x > -1$ and $q_i(x)$, $i = 1, \cdots, m$, are concave, we obtain

$$\begin{aligned}
F(x_\circ^{k-1}; \epsilon^k) &= c^T(x_\circ^{k-1} - x_\circ^k) + \epsilon^k \sum_{i=1}^m \ln \frac{q_i(x_\circ^k)}{q_i(x_\circ^{k-1})} \\
&\leq c^T(x_\circ^{k-1} - x_\circ^k) + \epsilon^k \sum_{i=1}^m \frac{q_i(x_\circ^k) - q_i(x_\circ^{k-1})}{q_i(x_\circ^{k-1})} \\
&\leq \epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x_\circ^{k-1})^T(x_\circ^{k-1} - x_\circ^k)}{q_i(x_\circ^{k-1})} - \epsilon^k \sum_{i=1}^m \frac{\nabla q_i(x_\circ^{k-1})^T(x_\circ^{k-1} - x_\circ^k)}{q_i(x_\circ^{k-1})} \\
&= \frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^m \frac{\nabla q_i(x_\circ^{k-1})^T(x_\circ^{k-1} - x_\circ^k)}{q_i(x_\circ^{k-1})} \\
&\leq \frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^m \frac{|\nabla q_i(x_\circ^{k-1})^T(x_\circ^{k-1} - x_\circ^k)|}{q_i(x_\circ^{k-1})} \\
&\leq \sigma\delta_3\epsilon^{k-1},
\end{aligned}$$

where the last inequality is a consequence of Corollary 3.7 and Lemma 2.2. □

3.3. Bounding $F(x^k; \epsilon^k)$. Using the bounds developed in the previous two sections, we show in Theorem 3.9 below that we can specify values for the parameters α, δ, σ such that the condition

$$F(x^k; \epsilon^k) \leq \alpha\epsilon^k$$

is always satisfied.

THEOREM 3.9. *Let* x^k *be generated by Algorithm* 2.1, *with* $\alpha = 0.005$, $\delta = 0.06$, *and* $\sigma = 0.007$. *Then*

$$F(x^k; \epsilon^k) \leq \alpha\epsilon^k \quad \forall k.$$

Proof. For $k = 0, F(x^0; \epsilon^0) \leq \alpha\epsilon^0$ is satisfied by assumption. Now, assuming that $F(x^{k-1}; \epsilon^{k-1}) \leq \alpha\epsilon^{k-1}$, we shall show that $F(x^k; \epsilon^k) \leq \alpha\epsilon^k$. From Lemma 3.1, we know that

$$F(x^k; \epsilon^k) = F(x^k; \epsilon^{k-1}) + F(x_\circ^{k-1}; \epsilon^k) + \frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^m \ln \frac{q_i(x^k)}{q_i(x_\circ^{k-1})}. \tag{3.16}$$

Since $0.005 = \alpha \leq (\delta_1^2/2) - \bar{K}(\delta_1)$ for $\delta_1 = 0.114$, it follows from the assumption that $F(x^{k-1}; \epsilon^{k-1}) \leq \alpha\epsilon^{k-1}$ and Theorem 3.4 that

$$F(x^k; \epsilon^{k-1}) \leq F(x^{k-1}; \epsilon^{k-1}) - \delta\sqrt{\epsilon^{k-1}}\sqrt{\delta_2 F(x^{k-1}; \epsilon^{k-1})} + \left(\frac{\delta^2}{2} + 2\sigma\delta + \bar{K}(\delta)\right)\epsilon^{k-1}$$

$$\leq \alpha\epsilon^{k-1} - 0.0541\sqrt{\alpha}\epsilon^{k-1} + 0.002834\epsilon^{k-1} \tag{3.17}$$

$$\leq 0.00402\epsilon^{k-1}.$$

For $\sigma = 0.007, \delta_3 = 0.0096$, hence from Theorem 3.8, we have that

$$F(x_\circ^{k-1}; \epsilon^k) \leq 0.00007\epsilon^{k-1}. \tag{3.18}$$

Moreover, since $0.00402 \leq (\tilde{\delta}^2/2) - \bar{K}(\tilde{\delta})$, for $\tilde{\delta} = 0.1002$, it follows from (3.17) and Lemma 2.6 that

$$x^k \in E(x_\circ^{k-1}, \tilde{\delta}).$$

Therefore, from Lemma 2.2 we have that

$$\frac{\sigma}{\sqrt{m}}\epsilon^{k-1}\sum_{i=1}^{m} \ln \frac{q_i(x^k)}{q_i(x_\circ^{k-1})} \leq \sigma\tilde{\delta}\epsilon^{k-1} \leq 0.0008\epsilon^{k-1}. \tag{3.19}$$

Substituting (3.17), (3.18), and (3.19) into (3.16), we get that

$$F(x^k; \epsilon^k) \leq 0.00489\epsilon^{k-1} \leq \frac{0.00489}{1-\sigma}\epsilon^k \leq 0.00493\epsilon^k \leq \alpha\epsilon^k.$$

Hence, by the induction, the theorem holds. □

4. Initialization and complexity of the algorithm. In Algorithm 2.1, it is assumed that we are given an initial feasible point x^0 and barrier parameter ϵ^0 such that $F(x^0; \epsilon^0) \leq \alpha\epsilon^0$. We now discuss one way of getting such an x^0 and ϵ^0. Specifically, we will show that by choosing ϵ^0 large enough we can use the initialization procedure in Mehrotra and Sun [11] to find such an x^0.

Let $g(x) = \sum_{i=1}^m \ln(q_i(x))$ and $\omega = \text{argmax}\,\{g(x) \mid x \in \text{int } S\}$. By using Mehrotra and Sun's [11] initialization procedure, we can obtain a point ξ, in $O(m^2n^3L)$ arithmetic operations, where L is the length of the input data for (QCP), such that

$$g(\omega) - g(\xi) \leq 0.002. \tag{4.1}$$

Since, for $x \in \text{int } S$, $f(x; \epsilon^0) \equiv c^T x - \epsilon^0 \sum_{i=1}^m \ln(q_i(x)) = c^T x - \epsilon^0 g(x)$ and $F(x; \epsilon^0) = f(x; \epsilon^0) - f(\omega^0; \epsilon^0)$, where $\omega^0 = \text{argmin } f(x; \epsilon^0)$, we have that

$$F(\xi; \epsilon^0) = f(\xi; \epsilon^0) - f(\omega^0; \epsilon^0) \tag{4.2}$$

$$= c^T(\xi - \omega^0) + \epsilon^0(g(\omega) - g(\xi) + g(\omega^0) - g(\omega)).$$

From the definition of ω, we have that $g(\omega^0) - g(\omega) \leq 0$, which when combined with (4.1) and (4.2) imply that

$$F(\xi; \epsilon^0) \leq c^T(\xi - \omega^0) + 0.002\epsilon^0 \tag{4.3}$$

$$\leq \left(\frac{|c^T(\xi - \omega^0)|}{\epsilon^0} + 0.002\right)\epsilon^0.$$

Hence, by choosing $x^0 = \xi$ and $\epsilon^0 \geq |c^T(x^0 - \omega^0)|/(\alpha - 0.002)$, for $\alpha > 0.002$, it follows from (4.3) that

$$F(x^0; \epsilon^0) \leq \alpha\epsilon^0.$$

We can now prove the main complexity result of the paper.

THEOREM 4.1. *Let Algorithm* 2.1, *with* $\alpha = 0.005$, $\delta = 0.06$, $\sigma = 0.007$, *and* $0 < \bar{\epsilon} \leq (\epsilon^0/m)$, *be applied to* (QCP) *starting from a point* $x^0 = \xi$ *satisfying* (4.1) *and with an initial penalty parameter* $\epsilon^0 = 2R\|c\|/0.003$. *Then the algorithm terminates in* $O(\sqrt{m}\ln(\epsilon^0/\bar{\epsilon}))$ *iterations with a feasible solution* x^k *which satisfies*

$$c^T x^k - c^T x^* \leq \bar{\epsilon}. \tag{4.4}$$

Proof. Since

$$\frac{|c^T(x^0 - \omega^0)|}{\alpha - 0.002} \leq \frac{\|c\| \cdot \|x^0 - \omega^0\|}{0.003} \leq \frac{2R\|c\|}{0.003} = \epsilon^0,$$

it follows from our analysis above that $F(x^0; \epsilon^0) \leq \alpha\epsilon^0$; hence x^0, ϵ^0, and α satisfy the initialization requirements of Algorithm 2.1. Hence from Theorem 3.9 $F(x^k; \epsilon^k) \leq \alpha\epsilon^k$ for all k. Also, since $0.005 = \alpha \leq (\delta_1^2/2) - \bar{K}(\delta_1)$ for $\delta_1 = 0.114 < \frac{1}{3}$, it follows from Lemma 2.6 that $x^k \in E(x_\circ^k, \delta_1)$, and therefore from Lemma 2.2 that

$$\sum_{i=1}^{m} \frac{|\nabla q_i(x_\circ^k)^T(x^k - x_\circ^k)|}{q_i(x_\circ^k)} \leq \delta_1\sqrt{m}. \tag{4.5}$$

Recalling from Lemma 2.1 that

$$c = \epsilon^k \sum_{i=1}^{m} \frac{\nabla q_i(x_\circ^k)}{q_i(x_\circ^k)}, \tag{4.6}$$

it follows from (4.5) that

$$\begin{aligned} c^T x^k - c^T x_\circ^k &= \epsilon^k \sum_{i=1}^{m} \frac{\nabla q_i(x_\circ^k)^T(x^k - x_\circ^k)}{q_i(x_\circ^k)} \\ &\leq \epsilon^k \sum_{i=1}^{m} \frac{|\nabla q_i(x_\circ^k)^T(x^k - x_\circ^k)|}{q_i(x_\circ^k)} \leq \delta_1\sqrt{m}\epsilon^k. \end{aligned} \tag{4.7}$$

Also, since $0 \leq q_i(x^*) = q_i(x_\circ^k) + \nabla q_i(x_\circ^k)^T(x^* - x_\circ^k) + \frac{1}{2}(x^* - x_\circ^k)^T G_i(x^* - x_\circ^k)$, it follows from (4.6) that

$$\begin{aligned} c^T x_\circ^k - c^T x^* &= \epsilon^k \sum_{i=1}^{m} \frac{\nabla q_i(x_\circ^k)^T(x_\circ^k - x^*)}{q_i(x_\circ^k)} \\ &= \epsilon^k \sum_{i=1}^{m} \frac{q_i(x_\circ^k) - q_i(x^*) + \frac{1}{2}(x^* - x_\circ^k)^T G_i(x^* - x_\circ^k)}{q_i(x_\circ^k)} \\ &\leq \epsilon^k \sum_{i=1}^{m} \frac{q_i(x_\circ^k)}{q_i(x_\circ^k)} = m\epsilon^k, \end{aligned} \tag{4.8}$$

where the inequality above is a consequence of the nonnegativity of $q_i(x^*)$ and the negative semidefiniteness of G_i, for all i. Combining (4.7) and (4.8) yields

$$c^T x^k - c^T x^* \leq (0.114\sqrt{m} + m)\epsilon^k.$$

Since $\ln \epsilon^k = \ln \epsilon^0 + k \ln(1 - (\sigma/\sqrt{m})) \leq \ln \epsilon^0 - k(\sigma/\sqrt{m})$, it follows that both ϵ^k and $c^T x^k - c^T x^*$ will be reduced below $\bar{\epsilon}$ in $O(\sqrt{m} \ln(\epsilon^0/\bar{\epsilon}))$ iterations. □

It immediately follows from Theorem 4.1 and the computational cost of each iteration that the total number of arithmetic operations required by Algorithm 2.1 is bounded by $O(\min\{m^{1.5}n^2 \ln(\epsilon^0/\bar{\epsilon}), m^{0.5}n^3 \ln(\epsilon^0/\bar{\epsilon})\})$.

5. Final remark. The approach presented here can be applied to the general convex programming. This is the subject of a forthcoming report.

REFERENCES

[1] M. Ben Daya and C. M. Shetty, *Polynomial barrier function algorithms for linear programming*, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 1987.

[2] ———, *Polynomial barrier function algorithms for convex quadratic programming*, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, 1988.

[3] S. C. Fang and J. R. Rajasekera, *Controlled perturbations for quadratically constrained quadratic programs*, Math. Programming, 36 (1986), pp. 276–289.

[4] A. V. Fiacco and G. P. McCormick, *Nonlinear programming: Sequential unconstrained minimization techniques*, John Wiley, New York, 1968.

[5] D. Goldfarb and S. Liu, *An $O(n^3L)$ primal interior point algorithm for convex quadratic programming*, Tech. Report, Department of Industrial Engineering and Operations Research, Columbia University, New York, 1988; Math. Programming, to appear.

[6] C. C. Gonzaga, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, in Progress in Mathematical Programming, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, pp. 1–28.

[7] F. Jarre, *On the convergence of the method of analytic centers when applied to convex quadratic programs*, manuscript, Institut für Angewandte Mathematik und Statistik, Universität Wurzburg, Am Hubland, 8700 Wurzburg, West Germany, 1987; Math. Programming, to appear.

[8] N. Karmarkar, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[9] M. Kojima, S. Mizuno, and A. Yoshise, *A primal-dual interior point method for linear programming*, in Progress in Mathematical Programming, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, pp. 29–47.

[10] S. Mehrotra and J. Sun, *An algorithm for convex quadratic programming that requires $O(n^{3.5}L)$ arithmetic operations*, Math. Oper. Res., 15 (1990), pp. 342-363.

[11] ———, *A method of analytic centers for quadratically constrained convex quadratic programs*, Tech. Report 88-01, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 1988.

[12] R. C. Monteiro and I. Adler, *Interior path following primal-dual algorithms. Part* I: *Linear programming*, Math. Programming, 44 (1989), pp. 27–41.

[13] ———, *Interior path following primal-dual algorithms. Part* II: *Convex quadratic programming*, Math. Programming, 44 (1989), pp. 43–66.

[14] J. Renegar, *A polynomial-time algorithm based on Newton's method for linear programming*, Math. Programming, 40 (1988), pp. 59–93.

[15] G. Sonnevend, *An "analytical centre" for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming*, Proc. 12th IFIP Conference on System Modelling and Optimization, Budapest 1985, Lecture Notes in Control and Information Sciences 84, Springer-Verlag, Berlin, 1986.

[16] P. Vaidya, *An algorithm for linear programming which requires $O(((m+n)n^2 + (m+n)^{1.5})L)$ arithmetic operations*, Math. Programming, 47 (1990), pp. 175–201.

[17] Y. YE AND E. TSE, *An extension of Karmarkar's projective algorithm for convex quadratic programming*, Math. Programming, 44 (1989), pp. 157–179.

[18] Y. YE, *Interior algorithms for linear quadratic and linearly constrained convex programming*, Ph.D. thesis, Department of Engineering–Economic Systems, Stanford University, Stanford, CA, 1987.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 268–279, May 1991

LARGE STEP PATH-FOLLOWING METHODS FOR LINEAR PROGRAMMING, PART I: BARRIER FUNCTION METHOD*

CLOVIS C. GONZAGA†

Abstract. The algorithm proposed in this paper is the classical logarithmic barrier function method with Newton–Raphson steps for the internal minimization of the penalized function. Polynomial behavior is obtained by stopping each internal cycle when the iterates approach the central trajectory. Each master iteration updates the penalty parameter by a constant factor, and the overall complexity bound depends on this factor: $O(nL)$ Newton iterations for an arbitrary constant, and $O(\sqrt{n}L)$ iterations for a constant dependent on $\sqrt{n}$.

Key words. interior point methods, linear programming, barrier function methods, path following methods

AMS(MOS) subject classification. 49D

1. Introduction.

This is the first of two papers in which we study large step path-following algorithms for linear programming. Path-following algorithms are based on the central trajectory for the problem, first studied by Sonnevend [20], Bayer and Lagarias [1], and Megiddo [12]. This trajectory is the common set of optimizers for several auxiliary functions based on the logarithmic barrier function, introduced in optimization by Frisch [3]. Each auxiliary function gives rise to a different parameterization $\rho \mapsto x(\rho)$ of the path. In this paper the auxiliary function is the classical logarithmic barrier penalized function; in Part II we study the potential function.

Path-following methods have two levels: a master algorithm updates the parameter value and calls an internal minimization algorithm that reduces the auxiliary function until it generates a point that satisfies a proximity criterion in relation to the desired point on the path. The distinction between short and large steps is related to the parameter variation at each master iteration: typical short steps are such that all intermediate points are near the trajectory; typical large steps use large parameter variations at the master level, and the intermediate iterates may wander quite far from the path. Typical short step algorithms advance carefully along the path, with a worst-case complexity of $O(\sqrt{n}L)$ internal iterations, where n is the number of variables and L is the bit length of the input data. Large step methods are bolder but loose in worst-case complexity (typically $O(nL)$ internal iterations).

The algorithm described in this paper is the classical barrier function method, with a proximity criterion used to stop the internal iterations after each update of the penalty parameter. The logarithmic barrier function was first used in optimization by Frisch [3], and was extensively studied by Fiacco and McCormick in their celebrated book [2]. The barrier function method as developed by them adds a penalty function $\mu p(\cdot)$ to the criterion $f(\cdot)$ of a constrained nonlinear programming problem, and performs a sequence of unconstrained minimizations of this penalized function $f_\mu(\cdot) = f(\cdot) + \mu p(\cdot)$ for decreasing values of the parameter μ.

A master iteration consists of an update of the penalty parameter and a call to the internal algorithm, responsible for the minimization of the penalized function. The traditional way of updating the parameter is by a constant factor, $\mu \leftarrow \mu/(1+\beta)$,

* Recieved by the editors January 10, 1990; accepted for publication (in revised form) December 5, 1990. This paper was first presented at the Second Asilomar Workshop on Progress in Mathematical Programming, Monterey, CA, February 5–7, 1990.

† COPPE, Federal University of Rio de Janeiro, C. Postal 68511, 21945 Rio de Janeiro, RJ, Brazil (clov@lncc.bitnet).

and one assumes that the internal minimization ends with an exact solution of the unconstrained penalized problem. Elegant proofs of asymptotic convergence were obtained for this scheme, with mild assumptions on the barrier functions (see Polak [16]).

The first study of implementable algorithms (i.e., without the assumption that the internal algorithm obtains an exact solution) was made by Polak, and later published in his book [16]. He stops each internal cycle whenever the norm of the gradient of the penalized function becomes smaller than a constant. This constant must decrease as the number of master iterations increases, to guarantee asymptotic convergence. Polak was actually using a proximity criterion to stop the internal iterations: an improvement of his stopping rule will make the algorithm polynomial for linear programming problems.

The connection between Karmarkar's algorithm [9] and barrier function methods was pointed out very soon after Polak by Gill et al. [4], but no clear method for updating the penalty parameter was proposed at that time. An answer to this question came at the end of 1986, when Renegar [17] obtained the first algorithm for linear programming with a complexity bound of $O(\sqrt{n}L)$ iterations using a method of centers approach to trace the central path. Then Gonzaga [7] obtained the same bound for the barrier function approach, using short steps for the penalty updates. The barrier function approach was simultaneously studied in a primal-dual setting by Kojima, Mizuno, and Yoshise [11], also using short steps.

Several studies were then made, either improving the proofs and presentation, or extending the results to quadratic programming and the linear complementarity problem (see [5], [8], [10], [13], [15], [14], [18]). All resulting algorithms solve the linear or quadratic programming problem in $O(\sqrt{n}L)$ iterations, and all are based on short steps. The picture is somewhat frustrating: either a method allows large steps, like Karmarkar's algorithm or the affine polynomial method in [6] and has a complexity bound of $O(nL)$ iterations, or the bound is lower but the steps are short and inefficient.

The short step barrier function method updates the penalty parameter by a factor $1+\nu/\sqrt{n}$, where ν is a small constant, usually $\nu = 0.05$. With these very small changes in the penalty parameter, and with an initial point nearly centered, we can show that the method generates a sequence of nearly centered points, closely following the central path. If the constant ν is increased, more than one Newton–Raphson step may be needed to approach the trajectory again, after updating the penalty parameter. This paper shows that the number of Newton–Raphson steps will then be bounded by a fixed number for each value of ν, and the complexity analysis still holds true.

The final result will be that the overall number of Newton–Raphson steps will be $O(\nu\sqrt{n}L)$. If ν is a constant, then the bound is $O(\sqrt{n}L)$. If $\nu = \beta\sqrt{n}$, then the bound is $O(nL)$. This case corresponds to an update of a factor $1+\beta$ for the penalty parameter, the common practice proposed, for instance, in Fiacco and McCormick [2]. This last case was studied independently by Roos and Vial [19], who arrived at the complexity $O(nL)$ by a very elegant proof.

2. Central points and the barrier function. We consider the following linear programming problem:

$$\begin{array}{lll} \text{minimize} & c'x & \\ \text{subject to} & Ax = b & x \geq 0, \end{array} \tag{1}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A$ is an $m \times n$ full-rank matrix, $(0 < m < n)$.

We assume that the feasible set

$$S = \{x \in \mathbb{R}^n \mid Ax = b, x \geq 0\}$$

of (1) is bounded, and that its relative interior S^o is nonempty. We also assume that an initial interior feasible point x^o is available. Additional assumptions about x^o will be made and commented on below.

The dual linear programming problem associated with (1) is

$$\begin{array}{lll} \text{maximize} & b'w & \\ \text{subject to} & A'w + z = c, & z \geq 0. \end{array} \tag{2}$$

It is well known that the duality gap is given by

$$x'z = c'x - b'w,$$

for any feasible x, w, z.

A symmetrical statement of the primal and dual problems can be obtained by eliminating the variables w from the dual problem by operating on the equation $A'w + z = c$. This equation states that the difference between z and c is orthogonal to the null space $\mathcal{N}(A)$ of A, and hence $z \in \mathbb{R}^n$ is dual feasible if and only if $Pz = Pc$, $z \geq 0$, where P is the projection matrix onto $\mathcal{N}(A)$.

The barrier function. We begin by listing some results on the logarithm function around 1.

LEMMA 2.1. *Let $\lambda \in (-1, 1)$ be given. Then*

$$\log(1 + \lambda) \leq \lambda, \tag{3}$$

$$\log(1 + \lambda) \geq \lambda - \frac{\lambda^2}{2} \frac{1}{1 - |\lambda|}, \tag{4}$$

$$\frac{d}{d\lambda} \log(1 + \lambda) = \frac{1}{1 + \lambda} = 1 - \lambda + \frac{\lambda^2}{1 + \lambda}. \tag{5}$$

Proof. The first relation is a consequence of the concavity of the logarithmic function. The second relation is well known, and the proof can be found in Karmarkar [9]. The last one is immediate. □

The properties in Lemma 2.1 can now be extended to the barrier function.

The barrier function $p : \mathbb{R}^n_{++} \to \mathbb{R}$ is defined by

$$p(x) = -\sum_{i=1}^{n} \log x_i,$$

and has derivatives

$$\nabla p(x) = -x^{-1}, \qquad \nabla p(e) = -e, \tag{6}$$

$$\nabla^2 p(x) = X^{-2}, \qquad \nabla^2 p(e) = I, \tag{7}$$

where x^{-1} represents the vector whose ith component is $1/x_i$, $X = \operatorname{diag}(x_1, \cdots, x_n)$, and $e = [1, \cdots, 1]'$.

Note that the derivatives of $p(\cdot)$ are very simple at e, which motivates us to study its properties near this point. It will be easy to extend the results to any positive point by a scaling operation.

Effect of scaling on the barrier function. Consider a positive diagonal matrix D. We have

$$p(Dx) = p(x) - \sum_{i=1}^{n} \log d_i.$$

Given two points $x, y > 0$,

$$p(Dy) - p(Dx) = p(y) - p(x), \tag{8}$$

and hence scaling operations do not affect *variations* of $p(\cdot)$.

Variation of the barrier function around e.

LEMMA 2.2. *Consider a vector $d \in \mathbb{R}^n$ such that $\|d\|_\infty < 1$. Then*

$$p(e+d) \geq \nabla p(e)'d = -e'd, \tag{9}$$

$$p(e+d) \leq \nabla p(e)'d + \frac{\|d\|^2}{2}\frac{1}{1-\|d\|_\infty} = -e'd + \frac{\|d\|^2}{2}\frac{1}{1-\|d\|_\infty}, \tag{10}$$

$$\nabla p(e+d) = \nabla p(e) + d + o(d), \tag{11}$$

where $o(d) \in \mathbb{R}^n$ is such that

$$\|o(d)\| \leq \frac{\|d\|^2}{1-\|d\|_\infty}.$$

Proof. We have

$$p(e+d) = -\sum_{i=1}^{n} \log(1+d_i).$$

Since $d_i \in (-1,1)$ by hypothesis, it is enough to extend the properties (3)–(5) by summing them over i, and taking the infinity norm in the denominators. The extensions are straightforward, and can be found in many references, including Karmarkar [9]. □

Extension of the results to the penalized function. The penalized functions for the linear programming problem and its dual are defined for each value $\alpha \in \mathbb{R}$ by

$$f_\alpha(x) = \alpha c'x + p(x), \qquad g_\alpha(z) = \alpha b'y - p(z). \tag{12}$$

Since the penalized functions differ from $p(\cdot)$ only by linear factors, the approximations obtained above can be extended to them simply by changing the linear term. We now repeat expressions (9)–(11) for the primal penalized function, with $\|h\| = 1$, $\lambda \in [0,1)$:

$$f_\alpha(e+\lambda h) \geq f_\alpha(e) + \lambda \nabla f_\alpha(e)'h, \tag{13}$$

$$f_\alpha(e+\lambda h) \leq f_\alpha(e) + \lambda \nabla f_\alpha(e)'h + \frac{\lambda^2}{2}\frac{1}{1-|\lambda|}, \tag{14}$$

$$\nabla f_\alpha(e+\lambda h) = \nabla f_\alpha(e) + \lambda h + o(\lambda h), \tag{15}$$

where $o(\lambda h) \in \mathbb{R}^n$ is such that

$$\|o(\lambda h)\| \leq \frac{\lambda^2}{1-\lambda\|h\|_\infty} \leq \frac{\lambda^2}{1-\lambda}.$$

Central points. The primal and dual central trajectories are the sets of central points defined for each $\alpha \geq 0$ by

$$x(\alpha) = \text{argmin } \{f_\alpha(x) \mid Ax = b, x > 0\},$$
$$z(\alpha) = \text{argmax } \{g_\alpha(z) \mid Pz = Pc, z > 0\}.$$

To each nonnegative penalty parameter, a primal central point $x(\alpha)$ and a dual central point $z(\alpha)$ are associated. The central trajectory was introduced by Bayer and Lagarias [1], and studied in several references, including [7]. Under our hypotheses the central points are well defined and unique for each nonnegative value of the penalty parameter.

The cost value $c'x(\alpha)$ decreases with α and tends to the optimal value when $\alpha \to \infty$. Similarly, the dual cost associated to $z(\alpha)$ increases with α toward the value of an optimal solution. Primal and dual central points are related by the following lemma.

LEMMA 2.3. *Given $\alpha \geq 0$, the primal and dual central points associated to α are related by*

$$z(\alpha) = \frac{x(\alpha)^{-1}}{\alpha}, \tag{16}$$

$$x(\alpha)'z(\alpha) = \frac{n}{\alpha}. \tag{17}$$

Proof. Let $\alpha \geq 0$ be given, and let $x(\alpha)$ be the primal central point. Since, for any dual feasible solution y, z,

$$x(\alpha)'z = c'x(\alpha) - b'y,$$

the dual penalized function can be written as

$$g_\alpha(z) = -\alpha x(\alpha)'z - p(z) - \alpha c'x(\alpha).$$

Since the last term is constant, the gradient is given by

$$\nabla g_\alpha(z) = -\alpha x(\alpha) + z^{-1}.$$

Substituting $z = x(\alpha)^{-1}/\alpha$, or $z^{-1} = \alpha x(\alpha)$, we obtain $\nabla g_\alpha(z) = 0$, proving the first relation. Now, calculating $x(\alpha)'z(\alpha)$, we obtain directly the second relation, completing the proof. □

Summarizing the results in this section, we conclude that to each nonnegative value of the penalty parameter α, we associate a pair of primal and dual central points $x(\alpha)$ and $z(\alpha)$ such that the primal cost decreases with α and the dual cost increases with α. The dual cost is given by

$$v(\alpha) = c'x(\alpha) - x(\alpha)'z(\alpha) = c'x(\alpha) - n/\alpha. \tag{18}$$

3. Nearly centered points. Path-following methods compute a sequence of points that are near the central trajectory by some proximity criterion. In this section we derive properties of such points.

A point will be considered as proximal to the central point $x(\alpha)$ when it is well within the region of quadratic convergence of Newton–Raphson's method for the minimization of the penalized function.

Notation. Given $\alpha > 0$ and $x \in S^o$, the Newton–Raphson step for the function $f_\alpha(\cdot)$ from x on S will be denoted by $h_N(x, \alpha)$. The projection matrix onto the null space of a matrix M will be denoted P_M, with $P = P_A$.

LEMMA 3.1. *The Newton–Raphson direction $h_N(x, \alpha)$ is given by*

(i) $h_N(e, \alpha) = -P_A \nabla f_\alpha(e) = -P_A(\alpha c - e)$.

(ii) $h_N(x, \alpha) = -XP_{AX}X\nabla f_\alpha(x) = -XP_{AX}(\alpha Xc - e)$.

Proof. (i) From e, the linear approximation of $\nabla f_\alpha(\cdot)$ is given by

$$\nabla f_\alpha(e + d) \approx \nabla f_\alpha(e) + \nabla^2 f_\alpha(e)d = \nabla f_\alpha(e) + Id,$$

using (7). The point that corresponds to the minimum of the quadratic approximation of $f_\alpha(\cdot)$ on $\mathcal{N}(A)$ satisfies

$$P(\nabla f_\alpha(e) + d) = 0.$$

Since $d \in \mathcal{N}(A)$, we get $P_A \nabla f_\alpha(e) + d = 0$, completing the proof of item (i).

To show item (ii), remember that the result of applying the Newton–Raphson step is not affected by a change of scale. The scaling operation $x = Xy$ takes the point x to e, and reduces the problem to the first case.

The direction in (ii) corresponds to

$$h_N(x, \alpha) = X\bar{h}_N(e, \alpha),$$

where $\bar{h}_N(e, \alpha)$ is computed as in (i) after scaling, completing the proof. □

Now we are ready to define the proximity criterion.

DEFINITION 3.2. Given $\alpha > 0$ and $x \in S^o$, the *proximity* of x in relation to $x(\alpha)$ is given by

$$\delta(x, \alpha) = \|X^{-1}h_N(x, \alpha)\|.$$

The point x will be considered *nearly centered* if $\delta(x, \alpha)$ is small, where small will normally mean $\delta(x, \alpha) \leq 0.1$.

The Newton–Raphson direction is scale-independent, in the sense that the point obtained by a Newton–Raphson step does not depend on the coordinate system used in the computation. The proximity measure describes the length of the Newton–Raphson step that would be obtained after a scaling that takes the present point to the vector of ones. From the point e, the Newton direction coincides with the projected gradient direction,

$$h_N(e, \alpha) = -P\nabla f_\alpha(e) = -\alpha Pc + Pe.$$

Note that whenever $\delta(x, \alpha) < 1$, the full Newton step leads to a new interior point.

The properties associated to the proximity concept are summarized in the next lemma.

LEMMA 3.3. *Consider a point $x \in S^o$ and let $\delta(x, \alpha)$ be its proximity to the central point $x(\alpha)$.*

(i) *Proximity and distance*: *If* $\delta(x,\alpha) \le 0.1$ *then* $\|X^{-1}(x - x(\alpha))\| \le 0.115$.
(ii) *Proximity and function values*: *If* $\delta(x,\alpha) \le 0.1$ *then* $f_\alpha(x) - f_\alpha(x(\alpha)) \le 0.012$.
(iii) *Proximity and cost*: *If* $\delta(x,\alpha) \le 0.1$ *then* $|c'x - c'x(\alpha)| \le 0.2\sqrt{n}/\alpha$.
(iv) *Guaranteed descent*: *If* $\delta(x,\alpha) \ge 0.1$ *then* $f_\alpha(y) \le f_\alpha(x) - 0.004$, *where*

$$y = x + \frac{0.1}{\delta(x,\alpha)} h_N(x,\alpha).$$

Proof. All statements are related to scale-invariant concepts, and can be shown after a scaling $x = Xy$ that takes x to e. The notation is simplified if we assume without loss of generality that such scaling has been done, and that $x = e$. With this assumption,

$$h_N(e,\alpha) \;=\; -P\nabla f_\alpha(e) \;=\; -\alpha Pc + Pe.$$

(i) Consider the normalized direction $h = (x(\alpha) - e)/\|x(\alpha) - e\|$, and let us study the function $\lambda \in [0,1) \mapsto f_\alpha(e + \lambda h)$.

This function is convex, and has a unique minimizer at $\bar\lambda = \|x(\alpha) - e\|$. We must prove that whenever $\delta(x,\alpha) \le 0.1$, $\bar\lambda \le 0.115$. The derivative of this function is

$$\frac{d}{d\lambda} f_\alpha(e + \lambda h) = h'\nabla f_\alpha(e + \lambda d).$$

It increases monotonically, crossing zero at $\bar\lambda$. Using (15) and keeping in mind that $\|h\| = 1$,

$$\begin{aligned}\frac{d}{d\lambda} f_\alpha(e + \lambda h) &\ge h'\nabla f_\alpha(e) + \lambda - \frac{\lambda^2}{1-\lambda} \\ &\ge -0.1 + \lambda - \frac{\lambda^2}{1-\lambda},\end{aligned}$$

since $h'\nabla f_\alpha(e) = h'P\nabla f_\alpha(e) \ge -\|h\|\delta(e,\alpha) \ge -0.1$.

Now, substituting $\lambda = 0.015$, we obtain $d/d\lambda f_\alpha(e + 0.015h) > 0$, proving that the derivative crosses zero at $\bar\lambda < 0.015$.

(ii) Using (13) with $\|h\| = 1$ and $\lambda \le 0.115$,

$$f_\alpha(e + \lambda h) \ge f_\alpha(e) - 0.1 \times 0.115.$$

(iii) Define $d = e - x(\alpha)$. By (i),

$$\|d\| \le 0.115.$$

We then have

$$d'\nabla f_\alpha(e) = d'P\nabla f_\alpha(e) \le 0.115\|P\nabla f_\alpha(e)\| \le 0.0115.$$

Substituting $\nabla f_\alpha(e) = \alpha c - e$,

$$\alpha c'd - e'd \le 0.0115.$$

But $e'd \le \|d\|_1 \le \sqrt{n}\,\|d\|$, and hence

$$\alpha c'd \le 0.115\sqrt{n} + 0.0115.$$

Finally, since $\sqrt{n} \geq 1$,

$$\alpha c'(e - x(\alpha)) \leq (0.115 + 0.0115)\sqrt{n} \leq 0.2\sqrt{n}.$$

(iv) Let

$$y = e + 0.1h, \qquad h = -\frac{P\nabla f_\alpha(e)}{\|P\nabla f_\alpha(e)\|}.$$

Then y is feasible. From (14),

$$f_\alpha(e + 0.1h) \leq f_\alpha(e) - 0.1\|P\nabla f_\alpha(e)\| + \frac{0.01}{1.8} \leq f_\alpha(e) - 0.004,$$

completing the proof. □

The lemma above provides the tools to study the complexity of the algorithm in the next section.

4. The algorithm. The algorithm is precisely the barrier function method as developed in Fiacco and McCormick [2], with a special criterion for stopping the internal iterations. A master algorithm updates the penalty parameter by a constant factor $(1+\theta)$, with $\theta = \nu/\sqrt{n}$. An internal algorithm then executes Newton–Raphson iterations to find an approximate minimizer for the penalized function. The internal algorithm uses as stopping criterion the proximity to the central point associated to the parameter.

The master algorithm stops when a point x^k is found such that $\delta(x^k, \alpha_k) \leq 0.1$ and $\alpha_k \geq 1.2\, n\, 2^L$. At such a point we have by Lemma 2.3 that

$$c'x(\alpha) - \hat{v} \leq \frac{n}{\alpha},$$

where $\hat{v}$ is the value of an optimal solution. Using Lemma 3.3 (iii),

$$c'x^k - \hat{v} \leq \frac{n + 0.2\sqrt{n}}{\alpha} \leq \frac{1.2\, n}{\alpha} \leq 2^{-L},$$

and an optimal solution can be found by purifying x^k.

The initial point. We shall assume that the algorithm starts with a parameter $\alpha_o > 2^{-L}$, and with a point x^o such that $\delta(x^o, \alpha_o) \leq 0.1$. This can be obtained by an initialization procedure (see [7]) or by starting the algorithm with internal iterations (centralization) from a point x^o such that $p(x^o) \leq O(\sqrt{n}L)$. In this last case, an initial centralization can be performed in $O(\sqrt{n}L)$ Newton–Raphson steps (see Vaidya [21]).

ALGORITHM 4.1. *Large step barrier: given $\theta = \nu/\sqrt{n}$ with $\nu > 0$ and given $x^o \in S^o$ and $\alpha_o \geq 0$, such that $\delta(x^o, \alpha_o) \leq 0.1$.*

$k := 0$.
REPEAT (Master iteration)
  Penalty: $\bar{\alpha} := (1 + \theta)\alpha_k$.
  $j := 0$, $y^o := x^k$.
  REPEAT (Inner iteration)
    Direction: compute $h := h_N(y^j, \bar{\alpha})$ and $\delta := \delta(y^j, \bar{\alpha})$.
    Search: solve approximately the linesearch problem

$$\bar{\lambda} := \operatorname{argmin} \{f_\alpha(y^j + \lambda h) \mid \lambda \geq 0\}.$$

New point: $y^{j+1} := y^j + \bar{\lambda} h$.
$j := j + 1$
UNTIL $\delta \leq 0.1$.
$x^{k+1} := y^j$.
$\alpha_{k+1} := \bar{\alpha}$.
$k := k + 1$.
UNTIL $\alpha_k > 1.2\, n\, 2^L$

The linesearch can be any procedure with $O(1)$ iterations that produces a result at least as good as the steplength used by Lemma 3.3(iv).

The algorithm was stated in its Newton–Raphson format. As we have seen above, the Newton–Raphson direction (for the specific case of the penalized function) coincides with the scaling-steepest descent direction, and the algorithm may be rewritten to use this last approach.

LEMMA 4.2. *The algorithm stops in no more than* $((1+\theta)/\theta)O(L)$ *master iterations.*

Proof. At the start of master iteration k,

$$\alpha_k = (1+\theta)^k \alpha_o \leq 1.2\, n\, 2^L.$$

Taking logarithms in base 2,

$$k \log(1+\theta) \leq L + \log(1.2\, n) - \log \alpha_o.$$

But since $\log(\cdot)$ is concave and $\theta > 0$,

$$\log(1+\theta) \geq \theta \frac{d}{d\theta} \log(1+\theta) = \frac{\theta}{1+\theta},$$

and it follows that

$$k \frac{\theta}{1+\theta} \leq L + \log(1.2\, n) - \log \alpha_o.$$

For $\alpha_o > 2^{-L}$, the right-hand side is of order L, and it follows that

$$k \leq \frac{1+\theta}{\theta} O(L),$$

completing the proof. □

Our next step is to prove that the number of internal iterations in each iteration of the master algorithm is bounded by a fixed number. This fact will lead us directly to polynomial bounds for the complete algorithm.

THEOREM 4.3. *Consider an iteration of the master algorithm, with* $\bar{\alpha} = (1+\theta)\alpha_k$, *and let* $\Delta = 0.004$. *The number* J *of internal steps satisfies*

$$J\Delta \leq \frac{\theta^2}{1+\theta} n + \theta\sqrt{n} + 1. \tag{19}$$

Proof. To simplify the notation, let $\alpha = \alpha_k$, $\bar{\alpha} = (1+\theta)\alpha$.

Using Lemma 3.3(iv),

$$f_{\bar{\alpha}}(y^J) \leq f_{\bar{\alpha}}(y^o) - J\Delta.$$

Since $\bar{\alpha} = (1+\theta)\alpha$, for any $y \in S^o$

$$f_{\bar{\alpha}}(y) = f_\alpha(y) + \theta\alpha c'y,$$

and it follows that

$$f_\alpha(y^J) + \theta\alpha c'y^J \leq f_\alpha(y^o) + \theta\alpha c'y^o - J\Delta,$$

or

$$J\Delta \leq \theta\alpha c'y^o - \theta\alpha c'y^J + f_\alpha(y^o) - f_\alpha(y^J). \tag{20}$$

But $y^o = x^k$ was nearly centered for α, that is, $\delta(y^o, \alpha) \leq 0.1$. Hence, from Lemma 3.3(ii),

$$f_\alpha(y^o) - f_\alpha(y^J) \leq 0.012 < 1.$$

Introducing this in (20),

$$J\Delta \leq \theta\alpha c'y^o - \theta\alpha c'y^J + 1,$$

or equivalently,

$$\frac{J\Delta - 1}{\theta} \leq \alpha c'y^o - \alpha c'y^J.$$

We can now use the fact that both y^o and y^J are nearly centered to deal with points on the central path and the duality gaps associated to them: since $\delta(y^o, \alpha) \leq 0.1$ and $\delta(y^J, \bar{\alpha}) \leq 0.1$, from Lemma 3.3(iii)

$$\frac{J\Delta - 1}{\theta} \leq \alpha\left(c'x(\alpha) + \frac{0.2\sqrt{n}}{\alpha}\right) - \alpha\left(c'x(\bar{\alpha}) - \frac{0.2\sqrt{n}}{\bar{\alpha}}\right).$$

Simplifying this expression and using the definition of $\bar{\alpha}$,

$$\frac{J\Delta - 1}{\theta} \leq \alpha c'x(\alpha) - \alpha c'x(\bar{\alpha}) + \left(1 + \frac{1}{1+\theta}\right) 0.2\sqrt{n}.$$

Now consider the dual costs associated with α and $\bar{\alpha}$, defined in (18):

$$v(\bar{\alpha}) - v(\alpha) \geq 0.$$

Adding the last two inequalities and noting that $0.2(1 + 1/(1+\theta)) \leq 1$,

$$\frac{J\Delta - 1}{\theta} \leq \alpha(c'x(\alpha) - v(\alpha)) - \alpha(c'x(\bar{\alpha}) - v(\bar{\alpha})) + \sqrt{n}.$$

Finally, we can use (18) to obtain

$$\frac{J\Delta - 1}{\theta} \leq n - \frac{1}{1+\theta}n + \sqrt{n},$$

or equivalently,

$$J\Delta - 1 \leq \frac{\theta^2}{1+\theta}n + \theta\sqrt{n},$$

completing the proof. □

The total number K of Newton–Raphson iterations in the complete algorithm is given by the product of the bounds in Lemma 4.2 and Theorem 4.3:

$$K\Delta \leq \left[n\theta + (1+\theta)\sqrt{n} + \frac{1+\theta}{\theta}\right] O(L).$$

Substituting $\theta = \nu/\sqrt{n}$,

$$K\Delta \leq \left[\nu\sqrt{n} + \nu + \sqrt{n} + \frac{\sqrt{n}+\nu}{\nu}\right] O(L). \tag{21}$$

There is little interest in studying the case where ν is too small. The short step path-following algorithm in [7] uses $\nu = 0.05$ and needs only one Newton–Raphson iteration per iteration of the master algorithm. For $\nu = O(1)$, $\nu > 0.05$, we obtain from (21),

$$K \leq O(\nu\sqrt{n}L). \tag{22}$$

We shall consider two cases:

(1) For an arbitrary $\nu = O(1)$, the right-hand side is $O(\sqrt{n}L)$, and this is the complexity bound for the algorithm.

(2) For an arbitrary $\theta = O(1)$, or equivalently $\nu = \theta\sqrt{n}$, substitution into (22) immediately gives $K = O(nL)$.

The influence of ν in the complexity bounds seems to indicate that the computational work increases as we go from short to large steps. But with large steps, the worst case is really worse than what we expect from the algorithm. The guaranteed descent Δ is associated to points near the trajectory, and grossly underestimates the actual descent that should be obtained at nonproximal points.

But even if we account for the better behavior far from the trajectory, the complexity bound will still be $O(\nu\sqrt{n}L)$. The proved worst-case behavior degrades when ν increases. Two things can be said about this: first, when we increase ν we become bolder, trying bigger jumps and hoping for a great economy in the number of iterations. If the trajectory bends very much, it may become difficult to regain the track, and we end by increasing the work; a more conservative updating scheme is safer, but always expensive. In worst-case studies the conservative approach will do better. The second remark is that as ν increases, the method becomes more and more similar to the affine-scaling algorithm at the beginning of each inner iteration loop. If the complexity bound did not degrade, this fact could be used to establish a low polynomial bound for the affine-scaling algorithm, a feat widely believed to be impossible.

REFERENCES

[1] D. Bayer and J. C. Lagarias, *The non-linear geometry of linear programming,* i. *affine and projective scaling trajectories,* ii. *legendre transform coordinates,* iii. *central trajectories,* preprints, AT&T Bell Laboratories, Murray Hill, NJ, 1986.

[2] A. Fiacco and G. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley, New York, 1968.

[3] K. R. Frisch, *The logarithmic potential method of convex programming*, memorandum, University Institute of Economics, Oslo, Norway, 1955.

[4] P. Gill, W. Murray, M. Saunders, J. Tomlin, and M. Wright, *On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method*, Math. Programming, 36 (1986), pp. 183–209.

[5] D. GOLDFARB AND S. LIU, *An $O(n^3L)$ primal interior point algorithm for convex quadratic programming*, manuscript, Department of Industrial Engineering and Operations Research, Columbia University, New York, 1988; Math. Programming, to appear.

[6] C. GONZAGA, *Polynomial affine algorithms for linear programming*, Internal report ES-141/88, Programa de Eng. de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brazil, 1988; Math. Programming, 49 (1990), pp. 7–21.

[7] ———, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 1.

[8] F. JARRE, *On the convergence of the method of analytic centers when applied to convex quadratic programs*, DFG-Report 35, Institut für Angewandte Mathematik und Statistik, Universität Würzburg, Würzburg, West Germany, 1987; Math. Programming, to appear.

[9] N. KARMARKAR, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[10] M. KOJIMA, S. MIZUNO, AND A. YOSHISE, *A polynomial-time algorithm for a class of linear complementarity problems*, Math. Programming, 44 (1989), pp. 1–26.

[11] ———, *A primal-dual interior point method for linear programming*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 2.

[12] N. MEGIDDO, *Pathways to the optimal set in linear programming*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 8.

[13] S. MEHROTRA AND J. SUN, *An algorithm for convex quadratic programming that requires $O(n^{3.5}L)$ arithmetic operations*, Tech. Report 87-24, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 1987.

[14] R. C. MONTEIRO AND I. ADLER, *Interior path-following primal-dual algorithms, part* I*: Linear programming*, Math. Programming, 44 (1989), pp. 27–41.

[15] ———, *Interior path-following primal-dual algorithms, part* II*: Convex quadratic programming*, Math. Programming, 44 (1989), pp. 43–66.

[16] E. POLAK, *Computational Methods in Optimization*, Academic Press, New York, 1971.

[17] J. RENEGAR, *A polynomial-time algorithm based on Newton's method for linear programming*, Math. Programming, 40 (1988), pp. 59–94.

[18] C. ROOS AND J.-P. VIAL, *A polynomial method of approximate centers for linear programming*, Report 88-68, Faculty of Technical Mathematics and Informatics, Technische Universiteit Delft, Delft, the Netherlands, 1988.

[19] ———, *Long steps with the logarithmic penalty barrier function in linear programming*, report, Faculty of Technical Mathematics and Informatics, Technische Universiteit Delft, Delft, the Netherlands, 1989.

[20] G. SONNEVEND, *An analytical centre for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming*, in Lecture Notes in Control and Information Sciences 84, Springer-Verlag, New York, 1985, pp. 866–876.

[21] P. VAIDYA, *A locally well-behaved potential function and a simple Newton-type method for finding the center of a polytope*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 5.

SIAM J. OPTIMIZATION
Vol. 1, No. 2, pp. 280–292, May 1991

008

LARGE STEP PATH-FOLLOWING METHODS FOR LINEAR PROGRAMMING, PART II: POTENTIAL REDUCTION METHOD*

CLOVIS C. GONZAGA†

Abstract. The algorithm proposed in this paper is the affine polynomial potential reduction method, with new procedures for updating the lower bounds for an optimal solution of the linear programming problem. A method is developed for updating the lower bounds by large steps, with strict control over the duality gaps associated with each iterate. Two algorithms are obtained by this approach: the first one has complexity $O(nL)$ iterations, and a very simple updating procedure; the second one updates the lower bounds at points very near the central trajectory and achieves a complexity of $O(\sqrt{n}L)$ iterations.

Key words. interior point methods, linear programming, potential reduction methods, Karmarkar's algorithm

AMS(MOS) subject classification. 49D

1. Introduction.

This is the second of two papers in which we study large step path-following algorithms for linear programming, the first based on penalty functions and this one based on the potential function. The approaches are very similar in both papers, but the complexity analyses are quite different. The potential function is not convex, and proximity to a central point is not so easy to characterize as for the penalized function. Although we tried to make this paper self-contained, the reader would benefit from reading §§1–3 of Part I.

The first potential reduction method, Karmarkar's algorithm [9], used a special formulation for the linear programming problem and started with the hypothesis that the value of an optimal solution was known. This assumption can be eliminated by using lower bounds to the value of an optimal solution, and by improving these bounds iteratively.

The first nice method for updating the lower bounds was developed by Todd and Burrell [14], based on duality arguments. A lower bound v for the optimal value $\hat{v}$ is used by the algorithm until a condition is satisfied, and then it is updated. This condition was also studied in Gonzaga [4] and amounts to $P\nabla f(e,v) > -e$, where $f(\cdot,\cdot)$ is Karmarkar's potential function computed at e, the vector of ones, after a scaling operation (the notation will be introduced in §2). Here (only) P stands for the projection onto the null space of the constraint matrix for the special format used in Karmarkar's algorithm. The point we want to make is that this criterion looks like a proximity measure from the central path, the set of minimizers of the potential function for varying v. It is, in fact, a very loose proximity criterion, and gives rise to a simple method for updating v: increase v until the proximity criterion is no longer satisfied. There is no way of controlling the improvement of the lower bound, and the method may result in short steps (although practical evidence seems to deny this fact).

Some extra flexibility in the manipulation of lower bounds can be obtained for the polynomial affine potential reduction algorithm [5]. This method works on the problem in standard formulation, and also departs from the assumption that the

* Recieved by the editors January 10, 1990; accepted for publication (in revised form) December 5, 1990. This paper was first presented at the Second Asilomar Workshop on Progress in Mathematical Programming, Monterey, CA, February 5–7, 1990.

† COPPE, Federal University of Rio de Janeiro, C. Postal 68511, 21945 Rio de Janeiro, RJ, Brazil (clov@lncc.bitnet).

optimal value is zero. The method uses the potential function

$$f(x,v) = q\log(c'x - v) - \sum_{i=1}^{n} \log x_i,$$

with $q \geq n+\sqrt{n}$. The main result is that at $x = e$, $\|P\nabla f(e,v)\| > 1$ whenever v is *not* a lower bound to $\hat{v}$. It follows that v can be increased whenever $\|P\nabla f(e,v)\| < 1$, until equality is obtained. This method of updating lower bounds was studied by Freund [3] and can be seen as similar to the one by Todd and Burrell, but uses a much stronger proximity criterion to the central point that minimizes $f(\cdot,v)$. The procedure is not efficient, since the projected gradient will always have a norm of one after an update, and this is very small in relation to what can be obtained, for instance, by Todd and Burrell's procedure. The variation in v will be small, characterizing a short step method, and the complexity is $O(nL)$ iterations.

The first potential reduction algorithm to achieve a complexity of $O(\sqrt{n}L)$ iterations was proposed by Ye [16], using the primal-dual potential function devised by Todd and Ye [15]. The method makes use of explicit dual variables, which are updated at points near the central path. The method as originally written by Ye generates short steps, but it can easily be modified to enlarge the steps (see Anstreicher and Bosch [1]) and obtain the same properties as we obtain in the present paper.

Ye's complexity analysis, based on properties of the primal-dual potential function on the central path, is extremely simple and elegant. Freund [3] wrote an algorithm that is very similar to Ye's method, showing that the lower bound updates can be computed from primal information only.

Influenced by Ye's analysis of the primal-dual potential function, Kojima, Mizuno, and Yoshise [10] developed a primal-dual algorithm that works effectively with primal and dual variables with no resort to proximity criteria, and achieves a complexity bound of $O(\sqrt{n}L)$ iterations without following the central path. Their method performs at each iteration a primal-dual scaling that does not take the present iterate to the vector of ones. Recently, Gonzaga and Todd [7] devised a "primal or dual" algorithm with independent scaling for the primal and dual problems with similar properties.

This paper develops a new way of controlling the lower bounds in the affine potential reduction algorithm. As in other path-following algorithms, the proximity to the central path is used to update v, but now this is done in such a way as to reduce the duality gap by a predetermined arbitrary ratio. Two algorithms will be presented: the first one uses very loose proximity criteria and very flexible updates, and has a complexity of $O(nL)$ iterations; the second one is stricter, and has a complexity bound of $O(\sqrt{n}L)$ iterations.

It turns out that our method is also very similar to Ye's algorithm.[1] The main differences are in the predetermined gap reduction at each update, in the possibility of using higher values of q while keeping the complexity low, and mainly in the complexity analysis, done in the primal path-following framework.

2. The problem. We consider the linear programming problem:

$$\begin{array}{lll} \text{minimize} & c'x & \\ \text{subject to} & Ax = b, & x \geq 0, \end{array} \tag{1}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A$ is an $m \times n$ full-rank matrix, $(0 < m < n)$.

[1] This only became clear during the refereeing process.

We assume that the feasible set S is bounded and that a point x^o in its relative interior S^o is available.

Notation. The projection matrix onto $\mathcal{N}(A)$ will be denoted P. We also use the notation c_p and e_p for Pc and Pe, respectively.

The dual linear programming problem associated to (1) is

$$\begin{array}{ll} \text{maximize} & b'w \\ \text{subject to} & A'w + z = c, \qquad z \geq 0. \end{array} \tag{2}$$

The duality gap is given by $x'z = c'x - b'w$ for any feasible x, w, z, and (see Part I) $z \in \mathbb{R}^n$ is dual feasible if and only if $Pz = Pc$, $z \geq 0$.

The following functions are associated to the problem.

The barrier function is defined as

$$x \in \mathbb{R}^n_{++} \mapsto p(x) = -\sum_{i=1}^{n} \log x_i. \tag{3}$$

The penalized function is defined for each real α as

$$x \in \mathbb{R}^n_{++} \mapsto g_\alpha(x) = \alpha c'x + p(x). \tag{4}$$

The potential function is defined for each $v \leq \hat{v}$ as

$$x \in S^o \mapsto f_q(x, v) = q \log(c'x - v) + p(x), \tag{5}$$

where $q \geq n$ is a fixed number.

The barrier function and the penalized function are strictly convex in S^o, and the potential function, although not convex, has a unique minimizer for any value of v for $q \geq n$. This is a consequence of the strict convexity of the multiplicative potential function (the antilogarithm of the potential function), as was proved by Imai [8]. Besides this, $f_q(\cdot, v)$ is unimodal along any feasible direction.

We shall study the relationship between these functions. Since all algorithms will begin each iteration by a scaling operation that takes the present iterate to the point $e = [1, \cdots, 1]'$, it is sufficient to study differential properties of the functions at this point. It is well known that scaling operations do not affect the variations of the functions under study. We then have

$$\nabla p(e) = -e, \qquad \nabla^2 p(e) = I, \tag{6}$$

$$\nabla g_\alpha(e) = \alpha c - e, \qquad \nabla^2 g_\alpha(e) = I, \tag{7}$$

$$\nabla f_q(e, v) = \frac{q}{c'e - v} c - e, \qquad \nabla^2 f_q(e, v) = -\frac{q}{(c'e - v)^2} cc' + I. \tag{8}$$

The central path. The central trajectory, first studied for the linear programming problem by Sonnevend [13], Bayer and Lagarias [2], and Megiddo [11], can be defined in several equivalent ways. It is the set of minimizers of $g_\alpha(\cdot)$ for varying α, and equivalently the set of minimizers of $f_q(\cdot, v)$ for varying v, with $q \geq n$.

The point e is the central point associated to the penalty multiplier $\alpha \geq 0$ if and only if

$$P\nabla g_\alpha(e) = \alpha c_p - e_p = 0.$$

The point e is the central point associated to the lower bound v if and only if

$$P\nabla f_q(e, v) = \frac{q}{c'e - v} c_p - e_p = 0.$$

It is apparent that α and v define the same central point e if and only if

$$\alpha = \frac{q}{c'e - v}. \tag{9}$$

Primal-dual properties. Assume that e is the central point associated to α by the penalized function. Then $\alpha c_p = e_p$, and it follows that

$$P\frac{e}{\alpha} = c_p.$$

Hence $e/\alpha > 0$ is a feasible dual slack, and the duality gap is

$$z'e = e'\frac{e}{\alpha} = \frac{n}{\alpha}.$$

We conclude that if e is the central point associated to α, then $z = e/\alpha$ is a feasible dual slack, and the duality gap is given by n/α. The dual objective value is

$$v = c'e - \frac{n}{\alpha} \leq \hat{v}.$$

The conclusion is very interesting, and is central to our development: given a central point, this relationship immediately gives us a value for a lower bound.

To illustrate the usefulness of this fact, let us examine the potential function and assume that e is the central point associated to v:

(i) If $q = n$, then from (9),

$$\alpha = \frac{n}{c'e - v}$$

and v is the lower bound associated to α.

(ii) Assume now that $q > n$, and that the central point associated to v_1 has been (exactly) found. Then from (9),

$$\alpha = \frac{q}{c'e - v_1}.$$

By the analysis above, there exists $v_2 \leq \hat{v}$ such that

$$\alpha = \frac{n}{c'e - v_2}.$$

It follows that

$$\frac{c'e - v_2}{c'e - v_1} = \frac{n}{q}.$$

We conclude that after minimizing the potential function $f_q(\cdot, v_1)$, a new lower bound v_2 can be found such that v_2 is the dual objective value at a dual-feasible point, and the duality gap is reduced by the ratio n/q.

(iii) A third way of using the relationship will be useful in the future. Consider the numbers $q > r \geq n$, and assume that e is the minimizer of $f_q(\cdot, v_1)$. Then we can write

$$\alpha = \frac{q}{c'e - v_1} = \frac{r}{c'e - v_2},$$

that is, there exists a lower bound v_2 such that e minimizes $f_r(\cdot, v_2)$ on S^o. Again, this defines a new lower bound v_2 such that

$$\frac{c'e - v_2}{c'e - v_1} = \frac{r}{q}.$$

A conceptual algorithm. If we assume that exact minimizations can be easily computed, then the following algorithm will follow the central path from a given initial lower bound v_o.

ALGORITHM 2.1. *Conceptual path-following: given $q > n$, $v_o < \hat{v}$.*

$k := 0$.
REPEAT
  Find $x^{k+1} = \operatorname{argmin} \{f_q(x, v_k) \mid x \in S^o\}$.
  $v_{k+1} := c'x^{k+1} - \frac{n}{q}(c'x^{k+1} - v_k)$.
  $k := k + 1$.
UNTIL $c'x^k - v_k < 2^{-L}$.

3. Lower bounds at nearly central points. In Part I, we defined the proximity criterion that characterizes nearly central points. At $x = e$, the proximity measure for a given α is given by $\delta(e, \alpha) = \|\nabla g_\alpha(e)\|$.

Now we extend the primal-dual properties to points resulting from an imperfect minimization. This will provide tools to build methods like Algorithm 2.1 with implementable computations.

Initially, let us extend the results of the previous section by means of a lemma that seems to be interesting in itself: it provides a feasible dual slack for points near the central path. Note that the duality results are applicable to points satisfying $P\nabla g_\alpha(e) > -e$, a condition weaker than near centrality.

LEMMA 3.1. *For given $\alpha \geq 0$, assume that $P\nabla g_\alpha(e) > -e$. Then*

$$z = \frac{e + P\nabla g_\alpha(e)}{\alpha} \tag{10}$$

is a dual feasible slack, and the duality gap is

$$e'z \leq \frac{n}{\alpha}\left(1 + \frac{\|P\nabla g_\alpha(e)\|}{\sqrt{n}}\right). \tag{11}$$

Proof. Obviously $z > 0$. We must show that $Pz = c_p$:

$$Pz = \frac{e_p + \alpha c_p - e_p}{\alpha} = c_p.$$

The gap is given by

$$e'z = \frac{n + e'P\nabla g_\alpha(e)}{\alpha} \leq \frac{n + \|P\nabla g_\alpha(e)\|\sqrt{n}}{\alpha},$$

completing the proof. □

This lemma can be rephrased for the potential function.

LEMMA 3.2. *For given $q > n$ and $v_1 < \hat{v}$ assume that $P\nabla f_q(e, v_1) > -e$. Then*

$$z = \frac{e + P\nabla f_q(e, v_1)}{q}(c'e - v_1) \tag{12}$$

is a dual feasible slack, and the duality gap $c'e - v_2 = e'z$ *satisfies*

$$c'e - v_2 \leq \frac{n}{q}\left(1 + \frac{\|P\nabla f_q(e, v_1)\|}{\sqrt{n}}\right)(c'e - v_1). \tag{13}$$

Proof. The proof is immediate, by using $\alpha = q/(c'e - v_1)$. □

The updating rule. Let us rephrase the conclusions once more, to obtain clear rules for updating lower bounds. Now we shall impose the stricter proximity condition on $\|P\nabla f_q(e, v)\|$. Let $v_1 < \hat{v}$ be a known lower bound, and assume that (after scaling) $\|P\nabla f_q(e, v_1)\| \leq \epsilon$, where $q > n$ and $\epsilon \in (0, 1]$. Let r be such that

$$r \geq \left(1 + \frac{\epsilon}{\sqrt{n}}\right) n.$$

Then,

$$v_2 = c'e - \frac{r}{q}(c'e - v_1) \tag{14}$$

is a valid lower bound, and the following relations hold:

$$\frac{c'e - v_2}{c'e - v_1} = \frac{r}{q}, \tag{15}$$

$$\nabla f_q(e, v_1) = \nabla f_r(e, v_2). \tag{16}$$

The maximum increase in the lower bound is obtained with

$$r = \left(1 + \frac{\epsilon}{\sqrt{n}}\right) n,$$

and this will lead to an $O(nL)$-iteration algorithm in the next section. A more cautious increase in the lower bound will give us an $O(\sqrt{n}L)$-iteration algorithm in §5: there, we choose $q > r \geq 2n$, with $q - r = O(\sqrt{n})$.

4. $O(nL)$ **algorithms.** The algorithm in this section is an exact reproduction of the polynomial affine method in Gonzaga [5], with an added rule for updating the lower bounds. We shall save some space by merely indicating the technique for proving the complexity. Our main effort will be in clearing the process for updating the lower bounds.

Expression (14) provides the updating rule for the algorithm. Each iteration of the algorithm begins by a scaling that transports x^k to the point e. The scaling matrix is $X_k := \text{diag}(x_1^k, \cdots, x_n^k)$, and the scaled problem is obtained by the transformation $x = X_k y$. We shall simplify the presentation by keeping the same notation after scaling. The scaling procedure is clearly carried on in [5].

ALGORITHM 4.1. *Large step potential path-following: given* $q > n + \sqrt{n}$, $\epsilon \in (0, 1]$, $v_o < \hat{v}$, $x^o \in S^o$.

$k := 0$.

REPEAT

Scaling: redefine the problem after scaling by $x^k = X_k y$. Compute the projected vectors c_p and e_p.

Direction: $h := -P\nabla f_q(e, v_k) = -\dfrac{q}{c'e - v_k} c_p + e_p$.

If $\|h\| < \epsilon$ (or alternatively $\|h\|_\infty < \epsilon$) then

$$r := n\left(1 + \frac{\|h\|}{\sqrt{n}}\right),$$
$$v_{k+1} := c'e - \frac{r}{q}(c'e - v_k),$$
$$h := -\frac{q}{c'e - v_{k+1}}c_p + e_p.$$

Else $v_{k+1} := v_k$.
Linesearch: find an approximate solution for

$$\bar{\lambda} := \operatorname{argmin}\{f_q(e + \lambda h, v_{k+1}) \mid \lambda > 0, e + \lambda h > 0\}.$$

$y := e + \bar{\lambda}h$.
Scaling: $x^{k+1} := X_k y$.
$k := k + 1$.
UNTIL $c'x^k - v_k < 2^{-L}$.

The linesearch procedure can be any method with $O(1)$ iterations that produces a descent at least as good as the descent guaranteed by the theoretical results used in the complexity analysis below.

In the algorithm above, we used as proximity criterion either $\|h\| < \epsilon$ or $\|h\|_\infty < \epsilon$. In the first case, a sound decrease in duality gap at each update of v is guaranteed by $q > n + \sqrt{n}$. In fact, when v_k is updated we get

$$\frac{c'e - v_{k+1}}{c'e - v_k} \leq \frac{n}{q}\left(1 + \frac{\epsilon}{\sqrt{n}}\right) < 1, \tag{17}$$

if $q > n + \sqrt{n}$.

In the second case, using a proximity based on the infinity norm, a decrease in duality gap will be guaranteed by using $q > (1 + \epsilon)n$.

A large step algorithm is obtained if we use, for instance, $q = 5n$, and either of the proximity criteria. If we want large steps, there seems to be no reason to use ϵ smaller than 1, for any norm.

Complexity. If the projected gradient satisfies $\|P\nabla f_q(e, v)\| \geq \epsilon > 0$, then the potential function decreases along h by at least a constant that depends only on ϵ. We shall prove this for a specific value of ϵ in the next section. For $\epsilon = 1$, we proved in [5] that

$$f_q(e + 0.3h, v) - f_q(e, v) < -0.1.$$

If the projected gradient is not large enough, then the lower bound is updated, and $\log(c'e - v)$ changes according to (17):

$$\log(c'e - v_{k+1}) - \log(c'e - v_k) \leq \log\left[\frac{n}{q}\left(1 + \frac{1}{\sqrt{n}}\right)\right].$$

It is easy to check that if $q/n \geq e^{\delta/q}(1 + 1/\sqrt{n})$, then

$$f_q(e, v_{k+1}) - f_q(e, v_k) < -\delta.$$

The condition above is very easily obtained: take, for instance, $q = 2n$ and $\delta = 0.5$ for $n \geq 2$.

We conclude that in any iteration the potential function decreases by at least a constant value (the minimum between 0.1 and δ). If $q = O(n)$, then this fact leads to a termination in no more than $O(nL)$ iterations, as is proved in [5].

Remark. The complexity analysis is still simpler if we use $q \geq n + \sqrt{n}$ and add to the algorithm the following procedure: after changing the lower bound, check whether now $\|P\nabla fq(e, v_{k+1})\| \geq 1$. In the event that this is not true, increase v_{k+1} until the norm is equal to 1. In this case the algorithm will have a complexity bound of $O(nL)$ iterations with any $q \geq n + \sqrt{n}$, as Freund proved in reference [3]: the update rule simply improves that method.

5. A low complexity algorithm. The complexity bound of $O(\sqrt{n}L)$ will be obtained for Algorithm 4.1 by limiting the improvement of the lower bound in each iteration. The complexity analysis is more involved, and will be carried out in detail.

The algorithm uses two numbers, q and $r = 2n$ such that $q > r$, and the update rule will reduce the gap by

$$\frac{c'e - v_{k+1}}{c'e - v_k} = \frac{r}{q}. \tag{18}$$

The choice $r = 2n$ will be very helpful, because it yields very well conditioned Hessians for the potential function near a central point. This gives consistency to the measurement of proximity through the projected gradients. The analysis can be adapted to use any value $r \geq 2n$ such that $r = O(n)$ instead of $r = 2n$, but this seems to have little interest.

Instead of repeating the algorithm, we simply indicate the particularizations.

ALGORITHM 5.1. *Large step potential path-following: given* $q > r = 2n$, $v_o < \hat{v}$, $x^o \in S^o$.

In Algorithm 4.1, set $\epsilon = 0.03$ and suppress the definition of r.

The lower bounds. Since $r = 2n > (1 + 0.03/\sqrt{n})n$, the lower bounds are well defined.

The rest of the paper will be dedicated to showing that if $q - r = O(\sqrt{n})$, and x^o is conveniently chosen, then the algorithm solves the linear programming problem in no more than $O(\sqrt{n}L)$ iterations.

Complexity analysis. We must now clarify the proximity criterion. We must show that for $q \geq 2n$ the condition $\|P\nabla f_q(e, v)\| < \epsilon$ really implies that e is near the central point $x_q(v)$ associated with v. This sort of analysis is easy for the penalized function, because its Hessian matrix at e is the identity matrix (see Part I, Lemma 3.3). In our case this is not true, and we must examine the conditioning of the Hessian, in Kantorovich tradition.

The choice of q and r. Here we make some comments that will not be used in the actual proofs, but that we believe to be very interesting.

Let $h \in \mathcal{N}(A)$ be such that $\|h\| = 1$, and assume that e is the central point associated to v. Then, using elementary properties of orthogonal projection,

$$\nabla f_q(e, v)'h = \frac{q}{c'e - v}c'h - e'h = 0,$$

or equivalently,

$$\frac{c'h}{c'e - v} = \frac{e'h}{q}. \tag{19}$$

From the expression for the Hessian matrix (8),

$$h'\nabla^2 f_q(e,v)h \;=\; -q\left(\frac{c'h}{c'e-v}\right)^2 + \|h\|^2 \;=\; 1 - q\left(\frac{c'h}{c'e-v}\right)^2 \;\leq\; 1.$$

Using (19), we obtain

$$h'\nabla^2 f_q(e,v)h \;=\; 1 - q\left(\frac{e'h}{q}\right)^2 \;\geq\; 1 - \frac{n-1}{q},$$

since $\|e'h\| \leq \sqrt{n-1}$, because h has at least one negative component for a bounded feasible set.

It follows that the Hessian is positive definite on $\mathcal{N}(A)$ for $q > n-1$. Some possible choices for q are:

$$\begin{array}{lll} q = n & \text{with} & h'\nabla^2 f_q(e,v)h \;\in\; [1/n\,,\ 1] \\ q = n+\sqrt{n} & \text{with} & h'\nabla^2 f_q(e,v)h \;\in\; [1/\sqrt{n}\,,\ 1] \\ q = 2n & \text{with} & h'\nabla^2 f_q(e,v)h \;\in\; [n+1/2n\,,\ 1]. \end{array}$$

In the last case, $q \geq 2n$, we have

$$h'\nabla^2 f_q(e,v)h \in [0.5,\ 1],$$

and the potential function is very well behaved near the central point. This will allow us to prove that in this case the condition that $\|P\nabla f_q(e,v)\|$ is small assures that e is near the central point; if this projection is large, then a good improvement in the value of $f(\cdot)$ can be expected from a steepest descent step.

Let us then derive the explicit relationship between proximity and distance to a central point. Part (i) of the next lemma is similar to Lemma 3.3(iv) in Part I, and is a standard result. Part (ii) is similar to Lemma 3.3(i) in Part I, and is new.

LEMMA 5.2. *Let $h \in \mathcal{N}(A)$ be a direction such that $\|h\| = 1$, and assume that $q \geq 2n$. Then*

(i) (*guaranteed descent*) : *If $\nabla f_q(e,v)'h \leq -0.03$ then $f_q(e+0.04h, v) \leq f_q(e,v) - 0.00024$.*

(ii) (*maximum descent*) : *If $\|\nabla f_q(e,v)\| \leq 0.03$ then $f_q(e,v) - f_q(x_q(v),v) \leq 0.0011$.*

Proof. Initially, let us recall some differential properties of the potential function. For any $\lambda \in (-1,1)$,

$$\begin{aligned} &\log(1+\lambda) \leq \lambda, \\ &\log(1+\lambda) \leq \lambda - \frac{\lambda^2}{2} + \frac{\lambda^3}{3}, \\ &\log(1+\lambda) \geq \lambda - \frac{\lambda^2}{2}\frac{1}{1-|\lambda|}. \end{aligned}$$

These relations are obtained from the Taylor series for the logarithm around 1. The third inequality was used by Karmarkar. Adding these inequalities and using obvious simplifications (see, for instance, Karmarkar [9] or Part I), the results are extended to the barrier function:

$$p(e+\lambda h) \geq -\lambda e'h \;=\; \lambda\nabla p(e)'h, \tag{20}$$

$$p(e+\lambda h) \geq \lambda \nabla p(e)'h + \frac{\lambda^2}{2} - \frac{\lambda^3}{3}, \tag{21}$$

$$p(e+\lambda h) \leq \lambda \nabla p(e)'h + \frac{\lambda^2}{2}\frac{1}{1-|\lambda|}. \tag{22}$$

Now consider the first term of the potential function,

$$f_1(x) = q \log(c'x - v).$$

This can be rewritten as

$$f_1(e+\lambda h) = q \log(c'e - v) + q \log\left(1 + \lambda \frac{c'h}{c'e - v}\right).$$

Defining now $a = c'h/(c'e - v)$ (note that $a = (1/q)\nabla f_1(e)'h$),

$$f_1(e+\lambda h) = f_1(e) + q \log(1 + \lambda a).$$

Using the properties of the logarithm,

$$f_1(e+\lambda h) \leq f_1(e) + \lambda \nabla f_1(e)'h, \tag{23}$$

$$f_1(e+\lambda h) \geq f_1(e) + \lambda \nabla f_1(e)'h - q\frac{\lambda^2 a^2}{2}\frac{1}{1-|\lambda a|}. \tag{24}$$

We are ready to develop the approximations for the complete potential function and prove the lemma.

(i) (guaranteed descent): assume that $\nabla f_q(e,v)'h \leq -0.03$ along the normalized feasible direction h.

Adding the inequalities (23) and (22),

$$\begin{aligned} f_q(e+\lambda h, v) &= f_1(e+\lambda h) + p(e+\lambda h) \\ &\leq f_q(e,v) + \lambda \nabla f_q(e,v)'h + \frac{\lambda^2}{2}\frac{1}{1-|\lambda|} \\ &\leq f_q(e,v) - 0.03\lambda + \frac{\lambda^2}{2}\frac{1}{1-|\lambda|}. \end{aligned}$$

Substituting $\lambda = 0.04$ in this expression, we obtain the conclusion of the lemma, completing the proof of item (i).

(ii) (maximum descent): assume that $\|\nabla f_q(e,v)\| \leq 0.03$, and define the normalized direction to the central point,

$$h = \frac{x_q(v) - e}{\|x_q(v) - e\|}.$$

Then $x_q(v) = e + \lambda h$ for some $\lambda > 0$. Adding (24) and (21),

$$f_q(e+\lambda h, v) \geq f_q(e,v) + \lambda \nabla f_q(e,v)'h + \frac{\lambda^2}{2} - \frac{\lambda^3}{3} - q\frac{\lambda^2 a^2}{2}\frac{1}{1-|\lambda a|}. \tag{25}$$

By definition of a,

$$\nabla f_q(e,v)'h = qa - e'h.$$

Using the assumptions and the definition of h, $|\nabla f_q(e,v)'h| \leq 0.03$. It follows that

$$q|a| \leq |e'h| + 0.03.$$

But the normalized direction h has at least one negative component (because S is compact), and hence $|e'h| \leq \sqrt{n-1}$. From the last inequality,

$$|a| \leq \frac{\sqrt{n-1}+0.03}{q}.$$

The right-hand side decreases with n, for $q = 2n$. Since $n \geq 2$, we get for $q \geq 2n$,

$$|a| \leq \frac{\sqrt{n-1}+0.03}{2n} \leq 0.26$$

and also

$$qa^2 \leq \frac{(\sqrt{n}+0.03)^2}{2n} \leq 0.52.$$

The bound is obtained by expanding the numerator and doing obvious simplifications. Introducing this into (25),

$$f_q(e+\lambda h, v) - f_q(e,v) \geq -0.03\lambda + \frac{\lambda^2}{2} - \frac{\lambda^3}{3} - 0.26\frac{\lambda^2}{1-0.26\lambda}.$$

Using elementary algebra, or by plotting the function, it is easy to see that the right-hand side $\rho(\lambda)$ assumes a local minimum at a point near $\lambda = 0.073$, it is positive for $\lambda = 0.18$, and for any $\lambda \in [0, 0.18]$, $\rho(\lambda) \geq -0.0012$.

The left-hand side is unimodal, as we have seen above, decreases at $\lambda = 0$, and is positive at $\lambda = 0.18$. We conclude that it must have an absolute minimum at some $\lambda \in [0, 0.18]$, satisfying

$$f_q(e+\lambda h, v) - f_q(e,v) \geq -0.0012,$$

completing the proof. □

We can finally prove the main result, showing the improvement in gap during each cycle of iterations between two lower bound updates. The proof of the complexity bound will then follow trivially.

In the complexity analysis we shall use $q = 2n + \nu\sqrt{n}$, where $\nu \geq 1$. Smaller values of ν can be used as well, but there is no interest in studying very short steps.

LEMMA 5.3. *Let $q = 2n + \nu\sqrt{n}$, where $\nu \geq 1$, and define $\delta = 0.00024$. Let k and $k+J$ be two consecutive iterations in which the lower bound is updated, $0 \leq j \leq J$. Then*

$$\log(c'x^{k+j} - v_{k+j}) \leq \log(c'x^k - v_k) - j\frac{\delta}{\nu\sqrt{n}}. \tag{26}$$

Proof. By construction, after scaling at x^k,

$$\|P\nabla f_q(e, v_k)\| \leq 0.03.$$

Again by construction (see (16)),

$$P\nabla f_r(e, v_{k+j}) = P\nabla f_q(e, v_k).$$

Hence,

$$\|P\nabla f_r(e, v_{k+j})\| \leq 0.03.$$

It follows from both items of Lemma 5.2 that

$$q\log(c'x^{k+j} - v_{k+j}) + p(x^{k+j}) \leq q\log(c'x^k - v_{k+j}) + p(x^k) - j\delta,$$
$$r\log(c'x^{k+j} - v_{k+j}) + p(x^{k+j}) \geq r\log(c'x^k - v_{k+j}) + p(x^k) - \Delta,$$

where $\Delta = 0.0011$. Subtracting the inequalities,

$$\nu\sqrt{n}\log(c'x^{k+j} - v_{k+j}) \leq \nu\sqrt{n}\log(c'x^k - v_{k+j}) + \Delta - j\delta. \tag{27}$$

From the bound update on iteration k, the gap changes by

$$\frac{c'x^k - v_{k+j}}{c'x^k - v_k} = \frac{r}{q} = \frac{2n}{2n + \nu\sqrt{n}}.$$

Taking logarithms and simplifying,

$$\log(c'x^k - v_{k+j}) = \log(c'x^k - v_k) - \log\left(1 + \frac{\nu}{2\sqrt{n}}\right).$$

Substituting into (27) and simplifying,

$$\log(c'x^{k+j} - v_{k+j}) - \log(c'x^k - v_k) \leq \frac{\Delta - j\delta}{\nu\sqrt{n}} - \log\left(1 + \frac{\nu}{2\sqrt{n}}\right).$$

It is easy to check that for $\nu \geq 1$ and $n \geq 2$, $\log(1 + \nu/2\sqrt{n}) \geq 0.4/\sqrt{n}$, and it follows that

$$\begin{aligned}\log(c'x^{k+j} - v_{k+j}) - \log(c'x^k - v_k) &\leq \frac{1}{\nu\sqrt{n}}(\Delta - 0.4\nu - j\delta) \\ &\leq -j\frac{\delta}{\nu\sqrt{n}},\end{aligned}$$

completing the proof. □

The convergence proof is now an application of this lemma. We need some hypothesis about the initial point x^o, and the simplest one is: assume that $x^o = e$ is nearly central, that is,

$$\|P\nabla f_q(e, v_o)\| \leq 0.03.$$

The problem can be manipulated to enforce this condition, as was done in references [6] or [12], for example. Another possible initialization consists in starting with any point such that $f_q(x^o, v_o) - f_q(x_q(v_o), v_o) \leq O(\sqrt{n}L)$. In this case a nearly central point will be found in $O(\sqrt{n}L)$ iterations, as an immediate consequence of Lemma 5.2.

LEMMA 5.4. *Suppose that Algorithm* 4.1 *is used from the initial point* $x^o = e$ *and that* $\|P\nabla f_q(e, v_o)\| \leq 0.03$. *Assume also that* $c'e - v_o < 2^L$, $r = 2n$, $q = r + \nu\sqrt{n}$, *with* $\nu \geq 1$. *Then the algorithm stops in no more than* $O(\nu\sqrt{n}L)$ *iterations.*

Proof. Under the hypotheses, the lower bound is updated in the first iteration. Applying Lemma 5.3 recursively for consecutive sequences of iterations with fixed lower bound, we obtain for any iteration k

$$\log(c'x^k - v_k) \leq \log(c'x^o - v_o) - k\frac{\delta}{\nu\sqrt{n}}.$$

It is sufficient to take $k \geq 2\nu\sqrt{n}L/\delta$, to see that

$$\log(c'x^k - v_k) \;\leq\; \log(c'x^o - v_o) - 2L \;\leq\; -L,$$

completing the proof. □

REFERENCES

[1] K. ANSTREICHER AND R. BOSCH, *Long steps in a $O(n^3L)$ algorithm for linear programming*, manuscript, Yale School of Organization and Management, Yale University, New Haven, CT, 1989; Math. Programming, to appear.

[2] D. BAYER AND J. C. LAGARIAS, *The non-linear geometry of linear programming*, i. *affine and projective scaling trajectories*, ii. *legendre transform coordinates*, iii. *central trajectories*, preprints, AT&T Bell Laboratories, Murray Hill, NJ, 1986.

[3] R. M. FREUND, *Polynomial-time algorithms for linear programming based only on primal scaling and projected gradients of a potential function*, manuscript, Sloan School of Management, Massachussetts Institute of Technology, Cambridge, MA, 1988; Math. Programming, to appear.

[4] C. GONZAGA, *Conical projection algorithms for linear programming*, Math. Programming, 43 (1988), pp. 151–173.

[5] ———, *Polynomial affine algorithms for linear programming*, Internal report ES-141/88, Programa de Eng. de Sistemas e Computação, COPPE/UFRJ, Rio de Janeiro, Brazil, 1988; Math. Programming, 49 (1990), pp. 7–21.

[6] ———, *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 1.

[7] C. GONZAGA AND M. J. TODD, *An $O(\sqrt{n}L)$-iteration large-step primal-dual affine algorithm for linear programming*, Tech. Report 862, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1989.

[8] H. IMAI, *On the convexity of the multiplicative version of Karmarkar's potential function*, Math. Programming, 40 (1988), pp. 29–32.

[9] N. KARMARKAR, *A new polynomial time algorithm for linear programming*, Combinatorica, 4 (1984), pp. 373–395.

[10] M. KOJIMA, S. MIZUNO, AND A. YOSHISE, *An $O(\sqrt{n}L)$ iteration potential reduction algorithm for linear complementarity problems*, Res. Report B–217, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan, 1988; Math. Programming, to appear.

[11] N. MEGIDDO, *Pathways to the optimal set in linear programming*, in Progress in Mathematical Programming—Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, 1989, Chap. 8.

[12] R. C. MONTEIRO AND I. ADLER, *Interior path-following primal-dual algorithms, part* I*: Linear programming*, Math. Programming, 44 (1989), pp. 27–41.

[13] G. SONNEVEND, *An analytical centre for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming*, in Lecture Notes in Control and Information Sciences 84, Springer-Verlag, New York, 1985, pp. 866–876.

[14] M. TODD AND B. BURRELL, *An extension of Karmarkar's algorithm for linear programming using dual variables*, Algorithmica, 1 (1986), pp. 409–424.

[15] M. J. TODD AND Y. YE, *A centered projective algorithm for linear programming*, Math. Oper. Res., 15 (1990), pp. 508–529.

[16] Y. YE, *An $O(n^3L)$ potential reduction algorithm for linear programming*, manuscript, Department of Management Sciences, The University of Iowa, Iowa City, IA, 1988; Math. Programming, to appear.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 293-315, August 1991

TENSOR METHODS FOR UNCONSTRAINED OPTIMIZATION USING SECOND DERIVATIVES*

ROBERT B. SCHNABEL† AND TA-TUNG CHOW†

Abstract. A new type of method for unconstrained optimization, called a tensor method, is introduced. It is related in its basic philosophy to the tensor methods for nonlinear equations for Schnabel and Frank [*SIAM J. Numer. Anal.*, 21 (1984), pp. 815-843], but beyond that the methods have significant differences. The tensor method for unconstrained optimization bases each iteration upon a fourth order model of the objective function. This model consists of the quadratic portion of the Taylor series, plus low-rank third and fourth order terms that cause the model to interpolate already calculated function and gradient values from one or more previous iterates. This paper also shows that the costs of forming, storing, and solving the tensor model are not significantly more than these costs for a standard method based upon a quadratic Taylor series model. Test results are presented for sets of problems where the Hessian at the minimizer is nonsingular, and where it is singular. On all the test sets, the tensor method solves considerably more problems than a comparable standard method. On problems solved by both methods, the tensor method requires about half as many iterations, and half as many function and derivative evaluations as the standard method, on the average.

Key words. unconstrained optimization, tensor method, higher order model, singular problems

AMS(MOS) subject classification. 65K05

1. Introduction.

This paper describes a new method, called a *tensor method*, for solving the unconstrained optimization problem

(1.1) given $f: R^n \to R$, find $x_* \in R^n$ such that $f(x_*) \leqq f(x)$ for all $x \in D$,

where D is some open set containing x_*. We assume that $f(x)$ is at least twice continuously differentiable, and that n is of moderate size, say $n < 100$. Our objective is to create a general purpose method that is more reliable and efficient than state-of-the-art methods for solving such problems, particularly in cases where the evaluation of $f(x)$ and its derivatives is expensive. We especially intend to improve upon the efficiency and reliability of standard methods on problems where $\nabla^2 f(x_*)$ is singular.

The distinguishing feature of our new method is that it bases each iteration upon a fourth order model of $f(x)$, as opposed to the standard quadratic model. The third and fourth order terms of this model have special, low-rank forms that make the costs of using the higher order model reasonable. In particular, in comparison to standard methods, the formation and use of the fourth order tensor model requires no additional function or derivative evaluations per iteration, only a small number of additional arithmetic operations per iteration, and only a very small amount of additional storage.

The tensor method approach was introduced by Schnabel and Frank [1984], [1987], who describe tensor methods for solving systems of nonlinear equations. Their methods base each iteration of an algorithm for solving $F(x) = 0$, where $F: R^n \to R^n$, upon the second order model

$$M_T(x_c + d) = F(x_c) + J_c \cdot d + \tfrac{1}{2} T_c \cdot d^2. \tag{1.2}$$

Here x_c is the current iterate, $J_c \in R^{n \times n}$ is the Jacobian matrix $F'(x_c)$ or an approximation to it, and $T_c \in R^{n \times n \times n}$ is a low-rank "tensor." In Schnabel and Frank's computational

* Received by the editors August 25, 1989; accepted for publication (in revised form) October 26, 1990. This research was supported by Army Research Office grant DAAL03-88-K-0086 and National Science Foundation grant CCR-8702403.

† Department of Computer Science, Campus Box 430, University of Colorado, Boulder, Colorado 80309.

experiments, the use of the tensor methods led to significant improvements in efficiency and reliability over state-of-the-art methods for nonlinear equations that are based upon standard, linear models. In the case when $F'(x_c)$ is available, the average reductions measured in function and derivative evaluations ranged from 20 percent to 60 percent, on both nonsingular and singular problems. Frank [1984] also proved that this derivative tensor method has a three-step, order 1.16 local convergence rate on problems where rank $(F'(x_*)) = n-1$, whereas standard methods are linearly convergent under these conditions.

The tensor method described in this paper is related to the methods of Schnabel and Frank in its basic philosophy, but it is not a straightforward generalization of their methods. In particular, it is *not* the application of the model (1.2) to the problem $\nabla f(x) = 0$. This would correspond to using a third order model of $f(x)$; as we have already stated, we use a fourth order model instead. To help motivate the basic differences, we first summarize some features of standard methods for unconstrained optimization.

Standard methods for solving small to moderate size unconstrained optimization problems base each iteration upon a quadratic model of $f(x)$ around the current iterate x_c,

$$m(x_c + d) = f(x_c) + g_c \cdot d + \tfrac{1}{2} H_c \cdot d^2, \tag{1.3}$$

where $d \in R^n$, $g_c \in R^n$ is $\nabla f(x_c)$ or a finite-difference approximation to it, and $H_c \in R^{n \times n}$. (We use the notation $g_c \cdot d$ for $g_c^T d$, and $H_c \cdot d^2$ for $d^T H_c d$, at the suggestion of a referee, to be consistent with the subsequent tensor notation, e.g., (1.5).) Such methods can be divided into two classes: those where H_c is $\nabla^2 f(x_c)$ or a finite-difference approximation to it, and those where H_c is a secant approximation to the Hessian formed solely from current and previous gradient values. In this paper, we will consider standard and tensor methods of the first type, where both $\nabla f(x_c)$ and $\nabla^2 f(x_c)$ are available analytically or by finite differences at each iteration. A subsequent paper will discuss tensor methods for unconstrained optimization that are based solely on function and gradient values.

The fundamental method for unconstrained optimization, Newton's method, is defined when $\nabla^2 f(x_c)$ is nonsingular. It consists of using $H_c = \nabla^2 f(x_c)$ in (1.3) and setting the next iterate x_+ so that $d = (x_+ - x_c)$ is the critical point of (1.3), i.e.,

$$x_+ = x_c - \nabla^2 f(x_c)^{-1} \nabla f(x_c). \tag{1.4}$$

The basic properties of Newton's method are well known. If $\nabla^2 f(x_*)$ is nonsingular at a local minimizer x_*, and the initial iterate is sufficiently close to x_*, then the sequence of iterates generated by (1.4) converges quadratically to x_*. If the initial iterate is not sufficiently close to x_*, then the iterates produced by Newton's method may not converge to x_*, but they may be made to converge through the use of line search or trust region strategies (see, e.g., Fletcher [1980]; Gill, Murray, and Wright [1981]; Dennis and Schnabel [1983]). The main costs of unconstrained optimization methods based upon Newton's method are: one evaluation of $\nabla^2 f(x)$, and one or more evaluations of $\nabla f(x)$ and $f(x)$, at each iteration; the solution of a symmetric $n \times n$ system of linear equations at each iteration, costing a small multiple of n^3 arithmetic operations; and the storage of a symmetric $n \times n$ matrix.

One shortcoming of standard unconstrained minimization methods is that they do not converge quickly if the Hessian at the minimizer, $\nabla^2 f(x_*)$, is singular. Griewank and Osborne [1983] have shown that in this case, the iterates produced by (1.4) generally are linearly convergent at best, even if $\nabla^2 f(x_c)$ is nonsingular at all the iterates.

Furthermore, the third derivatives do not supply information in the direction(s) where the second derivative matrix is lacking, since the necessary conditions for minimization show that at any minimizer x_* where $\nabla^2 f(x_*)$ is singular with null vector v, $\nabla^3 f(x_*) \cdot vvd$ must also be 0 for all $d \in R^n$. Thus, adding an approximation to $\nabla^3 f(x_c)$ alone will not lead to better-than-linear convergence for such problems. An approximation to the fourth derivative $\nabla^4 f(x_c)$ as well, or at least the quantity $\nabla^4 f(x_c) \cdot v^4$, is necessary to obtain better-than-linear convergence.

This need for fourth order information in order to obtain fast convergence on singular problems is one reason why we will use a fourth order model, rather than a third order model, in our tensor methods for optimization. Other reasons are that a third order model is unbounded below, even though it may have a local minimizer, and that the information that is readily available in an optimization algorithm, namely values of $f(x)$ and $\nabla f(x)$ at previous iterates, naturally supports the use of a fourth order tensor model. Note that these conditions are quite different from the situation for systems of nonlinear equations, where an approximation to $F''(x_*)$ (analogous to $\nabla^3 f(x_*)$) is sufficient to produce faster-than-linear convergence on problems where the Jacobian at the solution is singular, and where only one piece of interpolation information ($F(x)$, analogous to $\nabla f(x)$) is readily available from each previous iterate.

For these reasons, we have based the tensor methods for unconstrained minimization discussed in this paper upon the fourth order tensor model

$$m_T(x_c+d) = f(x_c) + \nabla f(x_c) \cdot d + \tfrac{1}{2}\nabla^2 f(x_c) \cdot d^2 + \tfrac{1}{6} T_c \cdot d^3 + \tfrac{1}{24} V_c \cdot d^4, \tag{1.5}$$

where by $\nabla f(x_c)$ and $\nabla^2 f(x_c)$ we mean either these analytic derivatives, or finite-difference approximations to them, and where $T_c \in R^{n \times n \times n}$ and $V_c \in R^{n \times n \times n \times n}$ are symmetric. (The symmetry of T_c and V_c is another significant difference between tensor models for unconstrained optimization and for nonlinear equations, where T_c is *not* symmetric with respect to its first index.) The three-dimensional object T_c and the four-dimensional object V_c are referred to as tensors, hence we call (1.5) a tensor model, and methods based on (1.5) tensor methods. Before proceeding, we define the notation concerning these tensors that is used above and in the remainder of this paper.

DEFINITION 1.1. Let $T \in R^{n \times n \times n}$. Then for $u, v, w \in R^n$, $Tuvw \in R$, $Tvw \in R^n$, with

$$T \cdot uvw = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} T[i,j,k]u[i]v[j]w[k],$$

$$(T \cdot vw)[i] = \sum_{j=1}^{n} \sum_{k=1}^{n} T[i,j,k]v[j]w[k], \qquad i = 1, \cdots, n.$$

DEFINITION 1.2. Let $V \in R^{n \times n \times n \times n}$. Then for $r, u, v, w \in R^n$, $Vruvw \in R$, $Vuvw \in R^n$ with

$$V \cdot ruvw = \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} V[i,j,k,l]r[i]u[j]v[k]w[l],$$

$$(V \cdot uvw)[i] = \sum_{j=1}^{n} \sum_{k=1}^{n} \sum_{l=1}^{n} V[i,j,k,l]u[j]v[k]w[l], \qquad i = 1, \cdots, n.$$

The obvious choices of T_c and V_c in (1.5) are $\nabla^3 f(x_c)$ and $\nabla^4 f(x_c)$; these would make (1.5) the first five terms of the Taylor series expansion of f around x_c. We will not consider using the actual higher order derivatives in the tensor model, however, because the cost of doing so would be prohibitive. In particular, $O(n^4)$ partial derivatives would have to be evaluated or approximated at each iteration; storing these

derivatives would take $O(n^4)$ locations; and finding a minimizer or the model would require the solution of a difficult minimization problem in n variables at each iteration. Each of these reasons alone is sufficient to reject this alternative for a general purpose method, although we note that, for some functions $f(x)$ with special forms, using analytic higher order information can be viable (see Jackson and McCormick [1986]).

Instead, our new method will choose T_c and V_c in (1.5) to be simple, low-rank symmetric approximations to $\nabla^3 f(x_c)$ and $\nabla^4 f(x_c)$ that are formed from previously calculated function and gradient values. The remainder of this paper will show how we efficiently form and solve such a tensor model, how we incorporate it into a complete unconstrained optimization algorithm, and what the computational performance of this algorithm is. Section 2 describes how we form the tensor model, and shows that this requires only a small multiple of n^2 additional arithmetic operations per iteration, and a small multiple of n additional storage locations. In § 3 we show how we solve this model using only $O(n^2)$ more operations per iteration than the $O(n^3)$ operations that are needed by the standard quadratic model. A full tensor algorithm for unconstrained optimization is presented in § 4. In § 5 we present test results of our tensor method on problems from Moré, Garbow, and Hillstrom [1981], and on modifications of these problems constructed so that $\nabla^2 f(x_*)$ is singular. We compare these results to those obtained from a state-of-the-art algorithm that uses the standard quadratic model (1.3), but is identical to the tensor algorithm in virtually all other respects. We briefly summarize our research and comment on possible extensions of this work in § 6.

We will denote members of a sequence of n-vectors x by $\{x_k\}$, where each $x_k \in R^n$, and to avoid ambiguity with this notation, we will continue to denote components of a vector $v \in R^n$ by $v[i] \in R$. We will also continue to abbreviate terms of the form *dd*, *ddd*, and *dddd* in our tensor models by d^2, d^3, and d^4, respectively.

2. Forming the tensor model. Now we discuss how we select the tensor terms $T_c \in R^{n\times n\times n}$ and $V_c \in R^{n\times n\times n\times n}$ in the model

$$m_T(x_c+d) = f(x_c) + \nabla f(x_c)\cdot d + \tfrac{1}{2}\nabla^2 f(x_c)\cdot d^2 + \tfrac{1}{6}T_c \cdot d^3 + \tfrac{1}{24}V_c \cdot d^4. \tag{2.1}$$

We have already stated that T_c and V_c will not contain actual third and fourth derivative information. Instead, we will use the third and fourth order terms in (2.1) to cause the model to interpolate function and gradient information that has been computed at some previous iterations of the optimization algorithm. In particular, we will select p not necessarily consecutive past iterates $x_{-1}, \cdots, x_{-p}$, and ask that the model (2.1) interpolate $f(x)$ and $\nabla f(x)$ at these points, i.e.,

$$f(x_{-k}) = f(x_c) + \nabla f(x_c)\cdot s_k + \tfrac{1}{2}\nabla^2 f(x_c)\cdot s_k^2 + \tfrac{1}{6}T_c \cdot s_k^3 + \tfrac{1}{24}V_c \cdot s_k^4, \qquad k = 1, \cdots, p \tag{2.2a}$$

$$\nabla f(x_{-k}) = \nabla f(x_c) + \nabla^2 f(x_c)\cdot s_k + \tfrac{1}{2}T_c \cdot s_k^2 + \tfrac{1}{6}V_c \cdot s_k^3, \qquad k = 1, \cdots, p \tag{2.2b}$$

where

$$s_k = x_{-k} - x_c, \qquad k = 1, \cdots, p. \tag{2.2c}$$

First, we briefly summarize how the past points $x_{-1}, \cdots, x_{-p}$ are selected. Then we discuss how we select T_c and V_c so that (2.2) is satisfied.

The set of past points used in the interpolation conditions (2.2) is selected by the procedure given in Schnabel and Frank [1984]. We always select the most recent previous iterate, and then select each preceding past iterate if the step from it to x_c makes an angle of at least θ degrees with the subspace spanned by the steps to the already selected, more recent iterates. Values of θ between 20 and 45 degrees have proven to be best in practice; therefore, the selected directions $\{s_k\}$ are strongly linearly

independent. This procedure is easily implemented using a modified Gram-Schmidt algorithm. We also set an upper bound

$$p \leqq n^{1/3} \tag{2.3}$$

on the number of past points. This bound was motivated by computational experience, which showed that using more than about $n^{1/3}$ interpolation conditions rarely helped much, and also by the desire to keep the storage and arithmetic costs of our tensor method low. In fact, however, our computational results will show that the strong linear independence criterion discussed above usually limits p far more severely than (2.3).

Now we will discuss how we choose T_c and V_c to satisfy (2.2). First we show that the interpolation conditions (2.2) uniquely determine $T_c \cdot s_k^3$ and $V_c \cdot s_k^4$ for each $k = 1, \cdots, p$. Multiplying (2.2b) by s_k gives

$$\nabla f(x_{-k}) \cdot s_k = \nabla f(x_c) \cdot s_k + \nabla^2 f(x_c) \cdot s_k^2 + \tfrac{1}{2} T_c \cdot s_k^3 + \tfrac{1}{6} V_c \cdot s_k^4, \qquad k = 1, \cdots, p. \tag{2.4}$$

Let the unknown quantities $\alpha, \beta \in R^p$ be defined by

$$\alpha[k] = T_c \cdot s_k^3, \tag{2.5a}$$

$$\beta[k] = V_c \cdot s_k^4, \tag{2.5b}$$

for $k = 1, \cdots, p$. Then from (2.2a) and (2.4), we have the following systems of two linear equations in two unknowns for each of the p pairs $\alpha[k]$ and $\beta[k]$:

$$\tfrac{1}{2}\alpha[k] + \tfrac{1}{6}\beta[k] = q_1[k], \tag{2.6a}$$

$$\tfrac{1}{6}\alpha[k] + \tfrac{1}{24}\beta[k] = q_2[k], \tag{2.6b}$$

where $q_1, q_2 \in R^p$ are defined by

$$q_1[k] = \nabla f(x_{-k}) \cdot s_k - \nabla f(x_c) \cdot s_k - \nabla^2 f(x_c) \cdot s_k^2,$$

$$q_2[k] = f(x_{-k}) - f(x_c) - \nabla f(x_c) \cdot s_k - \tfrac{1}{2}\nabla^2 f(x_c) \cdot s_k^2,$$

for $k = 1, \cdots, p$. The system (2.6) is nonsingular, so each $\alpha[k]$ and $\beta[k]$ is uniquely determined.

Thus for each k, our interpolation conditions (2.2) are equivalent to (2.5b) (with $\beta[k]$ determined as shown) and (2.2b). These are $2np$ linear equations in $O(n^4)$ unknowns, meaning that the choices of T_c and V_c are vastly underdetermined.

We have chosen to select T_c and V_c from among the infinite number of possibilities by first choosing the smallest symmetric V, in the Frobenius norm, for which

$$V_c \cdot s_k^4 = \beta[k], \qquad k = 1, \cdots p,$$

where $\beta[k]$ is calculated by (2.6). (The Frobenius norm of any matrix or tensor A, denoted $\|A\|_F$, is the square root of the sum of the squares of all the elements of A.) The rationale behind this choice is to use the smallest fourth order term consistent with the interpolation conditions, thereby modeling as much of the function and gradient information as possible with a third order model. This choice also will give us the necessary fourth order information in the singular case. We then substitute this value of V_c into (2.2b), obtaining

$$T_c \cdot s_k^2 = a_k, \qquad k = 1, \cdots, p, \tag{2.7a}$$

where

$$a_k = 2(\nabla f(x_{-k}) - \nabla f(x_c) - \nabla^2 f(x_c) \cdot s_k - \tfrac{1}{6} V \cdot s_k^3), \qquad k = 1, \cdots, p. \tag{2.7b}$$

This is a set of $np < n^{4/3}$ linear equations in n^3 unknowns $T_c[i, j, k]$, $1 \leqq i, j, k \leqq n$. Finally, we choose the smallest symmetric T_c, in the Frobenius norm, which satisfies the equations (2.7). The use of the minimum norm solution here is consistent with the tensor method for nonlinear equations, and will again be a key to the efficiency of our method because it will cause T_c and V_c to have low rank. Note that (2.7a) implies (2.5a), so that the result of this process will satisfy (2.5a), and hence (2.2a), as well.

The solutions to the minimum norm problems that determine V_c and T_c are given by Theorems 2.2 and 2.3. We note that deriving the minimum norm T_c is much more difficult than in the tensor method for nonlinear equations, because of the symmetry requirement. First we define three- and four-dimensional rank-one tensors, which will feature prominently in these theorems and in the remainder of this paper.

DEFINITION 2.1. Let $u, v, w, x \in R^n$. The tensor $T \in R^{n \times n \times n}$, for which $T[i, j, k] = u[i] \cdot v[j] \cdot w[k]$, $1 \leqq i, j, k \leqq n$, is called a third order rank-one tensor and will be denoted $T = u \otimes v \otimes w$. The tensor $V \in R^{n \times n \times n \times n}$, for which $V[i, j, k, l] = u[i] \cdot v[j] \cdot w[k] \cdot x[l]$, $1 \leqq i, j, k, l \leqq n$, is called a fourth order rank-one tensor and will be denoted $V = u \otimes v \otimes w \otimes x$.

THEOREM 2.2. *Let $p \leqq n$, let $s_k \in R^n$, $k = 1, \cdots, p$ with $\{s_k\}$ linearly independent, and let $\beta \in R^p$. Define $M \in R^{p \times p}$ by $M[i, j] = (s_i^T s_j)^4$, $1 \leqq i, j \leqq p$, and define $\gamma \in R^p$ by $\gamma = M^{-1}\beta$. Then the solution to*

$$\underset{V_c \in R^{n \times n \times n \times n}}{\text{minimize}} \quad \|V_c\|_F \quad \text{subject to } V_c \cdot s_k^4 = \beta[k], \quad k = 1, \cdots, p \quad \text{and} \quad V_c \quad \text{symmetric} \tag{2.8}$$

is

$$V_c = \sum_{k=1}^{p} \gamma[k](s_k \otimes s_k \otimes s_k \otimes s_k). \tag{2.9}$$

Proof. Let the vector $\hat{v} \in R^{n^4}$ be defined by $\hat{v}^T = (V_c[1, 1, 1, 1], V_c[1, 1, 1, 2], \cdots, V_c[1, 1, 1, n], V_c[1, 1, 2, 1], \cdots, V_c[1, 1, 2, n], \cdots, V_c[n, n, n, n])$. Also, let the matrix $\hat{S} \in R^{p \times n^4}$ be defined so that row k of $\hat{S}$ is equal to $((s_k[1])^4, (s_k[1])^3(s_k[2]), (s_k[1])^3(s_k[3]), \cdots, (s_k[1])^3(s_k[n]), \cdots, (s_k[n])^4)$, i.e., the same order of subscripts as in $\hat{v}$. Then (2.8) is equivalent to

$$\underset{\hat{v}}{\text{minimize}} \quad \|\hat{v}\|_2 \quad \text{subject to } \hat{S}\hat{v} = \beta \quad \text{and} \quad V_c \quad \text{symmetric},$$

where V_c is the original form of $\hat{v}$. Since $\{s_k\}$ are linearly independent, $\hat{S}$ has full row rank. The solution to

$$\underset{\hat{v}}{\text{minimize}} \quad \|\hat{v}\|_2 \quad \text{subject to } \hat{S}\hat{v} = \beta$$

is $\hat{v} = \hat{S}^T(\hat{S}\hat{S}^T)^{-1}\beta$. By straightforward algebra, $\hat{S}\hat{S}^T = M$. Thus $\hat{v} = \hat{S}^T\gamma$, which, by reversing the transformation from $\hat{v}$ back to V_c, is equivalent to (2.9). Since V_c is symmetric, it is the solution to (2.8). □

THEOREM 2.3. *Let $p \leqq n$, let $s_k \in R^n$, $k = 1, \cdots, p$ with $\{s_k\}$ linearly independent, and let $a_k \in R^n$, $k = 1, \cdots, p$. The solution to*

$$\underset{T_c \in R^{n \times n \times n}}{\text{minimize}} \quad \|T_c\|_F \quad \text{subject to } T_c \cdot s_i^2 = a_i, \quad i = 1, \cdots, p \quad \text{and} \quad T_c \quad \text{symmetric} \tag{2.10}$$

is

$$T_c = \sum_{k=1}^{p} (b_k \otimes s_k \otimes s_k + s_k \otimes b_k \otimes s_k + s_k \otimes s_k \otimes b_k), \tag{2.11}$$

where $b_k \in R^n$, $k = 1, \cdots, p$, *and* $\{b_k\}$ *is the unique set of vectors for which* (2.11) *satisfies* $T_c \cdot s_i^2 = a_i$, $i = 1, \cdots, p$.

Proof. First, we show that the constraint set in (2.10) is feasible. Let $t_i \in R^n$, $i = 1, \cdots, p$, obey

$$t_i^T s_j = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

for $j = 1, \cdots, p$. Since the $\{s_i\}$ are linearly independent, such vectors t_i are obtainable via a QR factorization of the matrix whose columns are the s_i. Then

$$T = \sum_{i=1}^{p} (t_i \otimes t_i \otimes a_i + t_i \otimes a_i \otimes t_i + a_i \otimes t_i \otimes t_i - 2(a_i^T s_i)(t_i \otimes t_i \otimes t_i))$$

is a feasible solution to (2.10).

Dennis and Schnabel [1979] show that if the constraints in (2.10) are satisfiable, then the set of tensors $T_i \in R^{n \times n \times n}$ generated by the procedure $T_0 = 0$, and for all $j = 0, 1, 2, \cdots$, T_{2j+1}, is the solution of

(2.12) minimize $\| T_{2j+1} - T_{2j} \|_F$ subject to $T_{2j+1} \cdot s_i^2 = a_i, \qquad i = 1, \cdots, p,$

and T_{2j+2} is the solution of

$$\text{minimize} \quad \| T_{2j+2} - T_{2j+1} \|_F \quad \text{subject to } T_{2j+2} \quad \text{is symmetric,}$$

has a limit which is the unique solution to (2.10). (This type of iterated projection process was first used in an optimization setting in Powell [1970].)

Next we show that this limit has the form (2.11) for some set of vectors $\{b_k\}$, by showing that each T_{2j} has this form. This is trivially true for T_0. Assume it is true for some fixed j, i.e.,

$$T_{2j} = \sum_{k=1}^{p} (u_k \otimes s_k \otimes s_k + s_k \otimes u_k \otimes s_k + s_k \otimes s_k \otimes u_k) \tag{2.13}$$

for some set of vectors $\{u_k\}$. Then from Schnabel and Frank [1984], the solution to (2.12) is

$$T_{2j+1} = T_{2j} + \sum_{k=1}^{p} (v_k \otimes s_k \otimes s_k)$$

for some set of vectors $\{v_k\}$. Thus

$$\begin{aligned} T_{2j+2} &= T_{2j} + \frac{1}{3} \sum_{k=1}^{p} (v_k \otimes s_k \otimes s_k + s_k \otimes v_k \otimes s_k + s_k \otimes s_k \otimes v_k) \\ &= \sum_{k=1}^{p} \left(\left(u_k + \frac{v_k}{3} \right) \otimes s_k \otimes s_k + s_k \otimes \left(u_k + \frac{v_k}{3} \right) \otimes s_k + s_k \otimes s_k \otimes \left(u_k + \frac{v_k}{3} \right) \right), \end{aligned}$$

which again has the form (2.13). Thus by induction the solution T_c to (2.10) must have the form (2.11) for some set of vectors $\{b_k\}$.

Finally, we show that the set of vectors $\{b_k\}$, for which T_c given by (2.11) satisfies

$$T_c \cdot s_i^2 = a_i, \qquad i = 1, \cdots, p, \tag{2.14}$$

is unique. This will mean that equations (2.11) and (2.14) uniquely determine the solution to (2.10). Substituting (2.11) into (2.14) gives a system of np linear equations in np unknowns, where the matrix is a function of the $\{s_k\}$, the unknowns are the elements of the $\{b_k\}$, and the right-hand side consists of the elements of the $\{a_k\}$.

Since we showed above that (2.10) is feasible for any $\{a_k\}$, the above derivation and the theory of Dennis and Schnabel [1979] imply that for any set $\{s_k\}$, this linear system has at least one solution for any right-hand side. Thus the linear system must be nonsingular and have a unique solution. This means that the set of vectors $\{b_k\}$ is uniquely determined and completes the proof. □

Theorems 2.2 and 2.3 show that T_c and V_c determined by the minimum norm problems (2.10) and (2.8) have rank $2p$ and p, respectively. This is the key to making the tensor model efficient to store and solve. However, while the proof of Theorem 2.3 shows constructively that there is a unique T_c of the form (2.11) that satisfies (2.10), it does not give an efficient algorithm for finding it, since the proof involves solving a system of np linear equations in np unknowns. We now present an efficient method for finding T_c.

Substituting (2.11) into (2.14) gives the equations

$$a_i = \left(\sum_{k=1}^{p} (b_k \otimes s_k \otimes s_k + s_k \otimes b_k \otimes s_k + s_k \otimes s_k \otimes b_k) \right) \cdot s_i^2$$

$$= \sum_{k=1}^{p} b_k (s_k^T s_i)^2 + 2 \sum_{k=1}^{p} s_k (s_k^T s_i)(b_k^T s_i),$$

$i = 1, \cdots, p$ in the unknowns $\{b_k\}$. We can write these equations in the matrix form

$$A = BN + 2SM, \tag{2.15}$$

where $A \in R^{n \times p}$, with column k of $A = a_k$, $B \in R^{n \times p}$, with column k of $B = b_k$, $S \in R^{n \times p}$, with column k of $S = s_k$, and N, $M \in R^{p \times p}$ with $N_{ij} = (s_i^T s_j)^2$ and $M_{ij} = (s_i^T s_j)(b_i^T s_j)$, $1 \leqq i, j \leqq p$. Note that B contains the unknowns, that M is a linear function of these unknowns, and that A, N, and S are known. Premultiplying both sides of (2.15) by S^T gives

$$[S^T A] = [S^T B]N + 2[S^T S]M. \tag{2.16}$$

Defining $x_{ij} = b_j^T s_i$, $1 \leqq i, j \leqq p$, we can rewrite (2.16) in the form of p^2 linear equations in the p^2 unknowns x_{ij}:

$$\begin{bmatrix} s_1^T a_1 \\ s_1^T a_2 \\ \vdots \\ s_p^T a_p \end{bmatrix} = \begin{bmatrix} N & & \\ & \ddots & \\ & & N \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{pp} \end{bmatrix} + 2 \begin{bmatrix} w_{11} & & & \\ & w_{12} & & \\ w_{p1} & & \ddots & w_{1p} \\ & & & w_{pp} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ \vdots \\ x_{pp} \end{bmatrix}, \tag{2.17}$$

where each w_{ij} in the second matrix of (2.17) is a p-element given by $w_{ij} = [(s_i^T s_1)(s_1^T s_j), (s_i^T s_2)(s_2^T s_j), \cdots, (s_i^T s_p)(s_p^T s_j)]^T$. The only unknowns in (2.17) are the x_{ij}, so we can solve (2.17) for x_{ij}, and then compute M by

$$M_{ij} = (s_i^T s_j)(b_i^T s_j) = (s_i^T s_j) x_{ji}.$$

Finally, from (2.15), we can compute B by

$$B = (A - 2SM)N^{-1}. \tag{2.18}$$

Note that N is symmetric and positive definite since the $\{s_k\}$ are linearly independent.

We conclude this section by summarizing the costs to form and store the tensor model. The dominant costs of the process for calculating T_c that is summarized in

equations (2.17) and (2.18) are np^2 each multiplications and additions for calculating S^TA, the same cost for calculating $S \cdot M$, the same cost again for the backsolves in (2.18), roughly $np^2/2$ each multiplications and additions for calculating $s_i^T s_j$ for $1 \leqq i \leqq j \leqq p$, and $p^6/3$ each multiplications and additions for solving the system (2.17), for a total of $(7/2)np^2 + p^6/3$ each multiplications and additions. Since $p \leqq n^{1/3}$, these are all $O(n^2)$ or less. The additional costs of forming V_c are negligible, at most $O(p^3)$. In addition, the cost of forming the interpolation equations (2.2) includes the multiplication $\nabla^2 f(x_c) \cdot s_k$ for $k = 1, \cdots, p$, which requires $n^2 p$ each multiplications and additions. This is generally the largest cost of forming the tensor model. The Gram-Schmidt process for selecting the $\{s_k\}$ requires about $n^{5/3}$ arithmetic operations if $n^{1/3}$ vectors are considered. In summary, only a small multiple of n^2 additional arithmetic operations are required to form the model. We will see in § 5 that usually $p = 1$, so that the total additional cost per iteration is usually $n^2 + n^{5/3} + O(n)$ each additions and multiplications per iteration.

The storage required for forming and storing the tensor model is also small. The tensor terms T_c and V_c themselves require only the storage of the vectors b_k and s_k, which takes $2np \leqq 2n^{4/3}$ storage locations. In addition, the model formation process requires at most $2n^{4/3}$ storage locations for storing $n^{1/3}$ each past iterates and their gradients, $np \leqq n^{4/3}$ storage locations for intermediate quantities in (2.18), and $p^4 \leqq n^{4/3}$ storage locations for the factorization in solving (2.17). Thus the total additional storage is at most $6n^{4/3}$.

3. Solving the tensor model. In § 2 we showed how to find a rank $2p$ tensor T_c of the form (2.11), and a rank p tensor V_c of the form (2.9), for which the tensor model (2.1) interpolates the function and gradient values at p ($\leqq n^{1/3}$) past iterates. Substituting these values of T_c and V_c into (2.1), the tensor model has the form

$$m_T(x_c + d) = f(x_c) + \nabla f(x_c) \cdot d + \tfrac{1}{2}\nabla^2 f(x_c) \cdot d^2 + \tfrac{1}{2}\sum_{k=1}^{p} (b_k^T d)(s_k^T d)^2 + \tfrac{1}{24}\sum_{k=1}^{p} \gamma[k](s_k^T d)^4. \tag{3.1}$$

In this section we show how to efficiently find a minimizer of this model. Although equation (3.1) is a fourth order polynomial in n variables, we will show how to reduce its minimization to the minimization of a fourth order polynomial in p variables plus a quadratic in $n - p$ variables. For conciseness, we use the notation $g = \nabla f(x_c)$ and $H = \nabla^2 f(x_c)$ for the remainder of this section.

Let $S \in R^{n \times p}$, where column k of S is s_k, and the $\{s_k\}$ are linearly independent. Also, let $Z \in R^{n \times (n-p)}$ and $W \in R^{n \times p}$ have full column rank and satisfy $Z^T S = 0$ and $W^T S = I$, respectively. (Z and W can be calculated through the QR factorization of S; the efficient implementation of the operations involving Z and W is discussed later in this section.) Then we can write d in (3.1) in the form

$$d = Wu + Zt, \tag{3.2}$$

where $u \in R^p$, $t \in R^{n-p}$. Substituting (3.2) into (3.1) gives

$$m_T(x_c + Wu + Zt) = f(x_c) + g^T Wu + g^T Zt + \tfrac{1}{2}u^T W^T H W u + u^T W^T H Z t + \tfrac{1}{2}t^T Z^T H Z t + \tfrac{1}{2}\sum_{k=1}^{p} u[k]^2 (b_k^T W u + b_k^T Z t) + \tfrac{1}{24}\sum_{k=1}^{p} \gamma[k] u[k]^4. \tag{3.3}$$

Equation (3.3) is a quadratic with respect to t. Therefore, for the tensor model to have a minimizer, Z^THZ must be positive definite and the derivative of the model with respect to t must be 0, i.e.,

$$Z^Tg + Z^THZt + Z^THW^Tu + \tfrac{1}{2}Z^T \sum_{i=1}^{p} b_i u[i]^2 = 0, \tag{3.4}$$

which yields

$$t = -(Z^THZ)^{-1}Z^T\left(g + HWu + \tfrac{1}{2}\sum_{i=1}^{p} b_i u[i]^2\right). \tag{3.5}$$

Thus, if Z^THZ is positive definite, substituting (3.5) into (3.3) reduces the problem of minimizing the tensor model to finding a minimizer of

$$\hat{m}_T(u) = f + g^TWu + \tfrac{1}{2}u^TW^THWu + \tfrac{1}{2}\sum_{i=1}^{p} u[i]^2(b_i^TWu) + \tfrac{1}{24}\sum_{i=1}^{p} \gamma[i]u[i]^4$$

$$-\frac{1}{2}\left(g + HWu + \tfrac{1}{2}\sum_{i=1}^{p} b_i u[i]^2\right)^T Z(Z^THZ)^{-1}Z^T\left(g + HWu + \tfrac{1}{2}\sum_{i=1}^{p} b_i u[i]^2\right), \tag{3.6}$$

which is a fourth degree polynomial in p variables. If (3.6) has a minimizer u_*, then the minimizer of the original tensor model (3.1) is given by $d_* = Wu_* + Zt_*$, where t_* is determined by setting $u = u_*$ in (3.5). Note that this process is well defined even if H is singular, as long as Z^THZ is nonsingular and positive definite. This is possible if and only if $\text{rank}(H) \geqq n - p$.

There are several possible difficulties with this process. First, (3.6) may have multiple minimizers. If $p = 1$, we can find the minimizers analytically, and if there are two, we choose the value of u_* that is in the same valley of the function $\hat{m}_T(u)$ as $u = 0$. This choice can be shown to guarantee that there is a (nonlinear) descent path from x_c to $x_c + d_*$ for the model $m_T(x_c + d)$. If $p > 1$, we minimize (3.6) with a standard unconstrained minimization code (starting from $u = 0$) and use the minimizer it returns. We have found that these procedures generally produce a desirable minimizer.

Second, the tensor model may not have a minimizer, either because Z^THZ is not positive definite, or because (3.6) has no minimizer when Z^THZ is positive definite. Finally, even if (3.6) has a minimizer d_*, $x_c + d_*$ may not be an acceptable next iterate. These difficulties are addressed by using a global strategy.

We have tried both line search and trust region global strategies in conjunction with our tensor method. The line search strategy we used is simple: if (3.6) has a minimizer d_* that is in a descent direction, but $x_c + d_*$ is not an acceptable next iterate, we set $x_+ = x_c + \lambda d_*$ for some $\lambda \in (0, 1]$ using a standard line search. If (3.6) has no minimizer, or d_* is not in a descent direction, we find the next iterate by using a line search algorithm based on the standard quadratic model (1.3). The tensor method based on this strategy has performed quite well (see § 5), but we find that about 40 percent of the iterations cannot use the tensor model. In order to make fuller use of the tensor model, we have also tried a trust region strategy, which is the method that we concentrate on in this paper.

The trust region method approximately solves the problem

$$\underset{d \in R^n}{\text{minimize}} \quad m_T(x_c + d) \quad \text{subject to } \|d\|_2 \leqq \delta, \tag{3.7}$$

where $\delta \in R$ is the trust radius that is adjusted at each iteration. This is a standard type of approach for unconstrained optimization; see, for example, Fletcher [1980] and Dennis and Schnabel [1983]. Efficient methods exist for solving the trust region problem with quadratic models (see, e.g., Moré and Sorensen [1983]) but it is quite difficult to extend them to the tensor model. For this reason, in order to test the trust region tensor method approach initially, we used a penalty method to solve (3.7). This means that we solve (3.7) by solving a sequence of unconstrained optimization problems of the form

$$\underset{d \in R^n}{\text{minimize}} \quad m_T(x_c + d) + \sigma(d_p^T d_p - \delta^2)^2 \tag{3.8}$$

for increasing positive values of the scalar σ. (The details of selecting σ are given in Chow [1989].) As in most trust region algorithms, we only solve (3.7) approximately; in our implementation we stop when a solution $d_*(\sigma)$ to (3.7) satisfies $\|d_*(\sigma)\| \in [0.95\delta, 1.05\delta]$. This means that σ does not grow unboundedly, and in practice a small number of problems of the form (3.8) are solved per iteration. The penalty approach is only intended for initial test purposes, because it increases the cost of each iteration considerably due to the cost of solving (3.8), although it does not increase the cost in function and derivative evaluations. We will see that our best results so far have been obtained when p is constrained to be 1 at each iteration; an efficient but complicated method for solving (3.7) in this case is given in Chow [1989].

Finally, we discuss the costs of solving the tensor model. The main additional calculations in finding a minimizer of the tensor model, in comparison to minimizing a standard quadratic model, are the calculations involving the matrices Z and W. These are performed by calculating the decomposition $S = Q \cdot R$, where $Q \in R^{n \times n}$ is an orthogonal matrix that is the product of p Householder transformations, and $R \in R^{n \times p}$ consists of an upper triangular matrix R_1 in its first p rows, and is 0 otherwise. (Q is not actually formed, rather the p n-vectors that determine the p Householder transformations are stored; see, e.g., Stewart [1970].) Also let $\hat{R} \in R^{n \times p}$ consist of $(R_1)^{-1}$ in its first p rows and 0 otherwise. Then $W = Q \cdot \hat{R}$, so for any $v \in R^n$, we can calculate $W^T v$ in $2np$ each multiplications and additions by applying the p Householder transformations for Q^T followed by $O(p^2)$ operations to apply $(R_1)^{-1}$. Similarly, $Z = Q \cdot \hat{I}$, where $\hat{I}$ is 0 in its first p rows and the identity matrix in its bottom $n-p$ rows. Thus for any $v \in R^n$ we can calculate $Z^T v$ in $2np$ each multiplications and additions by applying Q^T and then $\hat{I}^T$. Using these techniques, it is straightforward to verify that all the calculations in the tensor method that involve Z and W, as well as the QR decomposition of S, can be performed in $4n^2p + O(np^2)$ each multiplications and additions per iteration; the leading term comes from calculating HQ and then $Q^T HQ$.

The other costs of minimizing the tensor model are $(n-p)^3/6$ each multiplications and additions for the factorization of $Z^T HZ$, and the cost of minimizing the fourth order polynomial in p variables (3.6), which is negligible in comparison to the $O(n^3)$ cost, especially when $p = 1$. Thus the total cost of minimizing the tensor model is only a small multiple of $n^2 p$ operations more than the $n^3/6$ cost of finding a minimizer of a standard quadratic model. Since $p \leqq n^{1/3}$ and we will see that usually $p = 1$, this is a very small additional cost.

At many iterations, the tensor model has a minimizer that is accepted as the next iterate, so these are the only costs of solving the tensor model. If a global strategy is needed, then the line search described above can be implemented with about the same cost as for a standard quadratic model, since given the factorization of Z^THZ, we can also factor H using only $O(n^2p)$ additional operations. In the case $p=1$, the trust region strategy can also be implemented as efficiently as in the quadratic case, i.e., requiring the minimization of the tensor model at each inner iteration, by using the techniques in Chow [1989]. The penalty approach is more expensive but is only intended for test purposes.

4. The complete tensor method algorithm. An outline of the complete tensor method algorithm that we used in our computational tests is given in Algorithm 4.1. The remainder of this section comments on several aspects of this algorithm that have not yet been discussed.

ALGORITHM 4.1. An iteration of the tensor method. Given x_c, $f(x_c)$, δ_c:

1. Calculate $\nabla f(x_c)$ and decide whether to stop. If not:
2. Calculate $\nabla^2 f(x_c)$.
3. Select p past points to use in the tensor model from among the $n^{1/3}$ most recent past points.
4. Calculate the terms T_c and V_c in the tensor model, so that the tensor model interpolates $f(x)$ and $\nabla f(x)$ at all the points selected in step 3.
5. Find a potential acceptable next iterate $x_c + d_T$ and a potential new trust radius δ_T using the tensor model and a trust region strategy.
6. Find a potential acceptable next iterate $x_c + d_N$ and a potential new trust radius δ_N using the quadratic model and a trust region strategy.
7. If $f(x_c + d_T) \Leftarrow f(x_c + d_N)$
   then set $x_+ = x_c + d_T$ and $\delta_+ = \delta_T$
   else set $x_+ = x_c + d_N$ and $\delta_+ = \delta_N$.
8. Set $x_c = x_+$, $f(x_c) = f(x_+)$, $\delta_c = \delta_+$, go to step 1.

The most important feature of Algorithm 4.1 that has not been previously discussed is that at each iteration, we calculate a potential new iterate based on the quadratic model, as well as a potential new iterate based on the tensor model. This means that we perform a full global strategy using each model, resulting in two points $x_c + d_T$ and $x_c + d_N$, both of which give sufficient decrease in $f(x)$ to be acceptable as the next iterate. Then we choose the one with the lower function value as the next iterate. Even though this strategy increases the cost of each iteration by at least one function evaluation (since it is necessary to evaluate $f(x)$ at both $x_c + d_T$ and $x_c + d_N$, and maybe at some unsuccessful trial points, in the global strategies), we have found that this approach substantially improves the efficiency of our tensor method as measured in function and derivative evaluations, as well as in iterations. We have not yet found a way to achieve the same efficiency without requiring the use of both models at each iteration.

Finally, we discuss some details of the steps of Algorithm 4.1. In steps 1 and 2, the gradient and Hessian are approximated by finite differences using Algorithms A5.6.3 and A5.6.2 in Dennis and Schnabel [1983], respectively. The algorithm stops if $\|\nabla f(x_c)\|_2 \leqq 10^{-5}$ or $\|d_c\|_2 \leqq 10^{-10}$. Step 3 was discussed in § 2; 45 degrees is used for the angle θ mentioned there. The procedures for calculating T_c and V_c in step 4 also were discussed in § 2.

In step 5, we first determine whether the tensor model has an acceptable minimizer within the trust region and, if so, we select this point as the solution to step 5. Otherwise, we solve the trust region problem (3.7) by a penalty method, as discussed in § 3, resulting in a candidate step d. Then we decide whether to accept $x_c + d$ as the solution to step 5, update the trust radius, and possibly repeat this process until an acceptable point $x_c + d_T$ is found. In step 6, we follow the exact same procedure, except that we only use the first three terms of the model. The procedure for determining whether the candidate step is acceptable in these trust region algorithms, and for updating the trust region, is identical to Algorithm A6.4.5 in Dennis and Schnabel [1983], except that: (1) every occurrence of *initslope* is changed to $\Delta fpred$, where $\Delta fpred$ is the difference of the values of the model being used (tensor or quadratic) at the candidate point and at x_c; (2) steps (9c.1-2) of Algorithm A6.4.5 are replaced by setting $\Delta fpred$ to this same value.

5. Test results. We have tested the tensor algorithm described in § 4 on a variety of nonsingular and singular problems. We compared it to an algorithm that is identical except that the third and fourth order terms T_c and V_c are always zero. That is, the comparison algorithm is a finite-difference Newton's method whose global strategy is a trust region problem solved by a penalty method. (We prefer this type of a controlled comparison to a comparison with some different quadratic model code that would inevitably differ in many other aspects as well. We note that using a penalty method to solve the trust region problem approximately, rather than a procedure such as in Moré and Sorenson [1983], has very little effect on the number of iterations or function evaluations that the method requires.) In this section we summarize our test results. The details of our computational results are provided in the Appendix. All our computations were performed on a MIPS computer, using double precision arithmetic.

First we tested our algorithms on the set of unconstrained optimization problems in Moré, Garbow, and Hillstrom [1981]. All these problems except the Powell singular problem have $\nabla^2 f(x_*)$ nonsingular. The dimensions of the problems range from 2 to 30.

Then we created singular test problems by modifying the nonsingular test problems of Moré, Garbow, and Hillstrom [1981]. All of the unconstrained optimization test problems in that paper are obtained by taking a system of nonlinear equations

$$F(x)^T = (f_1(x), \cdots, f_m(x)) \tag{5.1}$$

where $m \geqq n$ and each $f_i : R^n \to R$, and setting

$$f(x) = F(x)^T F(x) = \sum_{i=1}^{m} f_i^2(x). \tag{5.2}$$

In most cases, $F(x) = 0$ at the minimizer x_*, and $F'(x_*)$ is nonsingular. In these cases, Schnabel and Frank [1984] showed how to create singular systems of nonlinear equations from (5.1), by forming

$$\hat{F}(x) = F(x) - F'(x_*)A(A^TA)^{-1}A^T(x - x_*), \tag{5.3}$$

where $A \in R^{n \times k}$ has full column rank with $1 \leqq k \leqq n$. Thus $\hat{F}(x_*) = 0$ and $\hat{F}'(x_*)$ has rank $n - k$. To create a singular unconstrained optimization problem, we simply define the function

$$\hat{f}(x) = \tfrac{1}{2}\hat{F}(x)^T\hat{F}(x). \tag{5.4}$$

From (5.4) and $\hat{F}(x_*)=0$, we have $\nabla\hat{f}(x_*)=\hat{F}'(x_*)^T\hat{F}(x_*)=0$. From

$$\hat{F}'(x_*)=F'(x_*)[I-A(A^TA)^{-1}A^T] \tag{5.5}$$

and

$$\nabla^2\hat{f}(x_*)=\hat{F}'(x_*)^T\hat{F}'(x_*)+\sum_{i=1}^{m} f_i(x_*)\nabla^2 f_i(x_*)=\hat{F}'(x_*)^T\hat{F}'(x_*), \tag{5.6}$$

we know that $\nabla^2\hat{f}(x_*)$ has rank $n-k$.

By using (5.3) and (5.4), we created two sets of singular problems, with $\nabla^2\hat{f}(x_*)$ having rank $n-1$ and $n-2$, by using

$$A\in R^{n\times 1}, \qquad A^T=(1,1,\cdots,1)$$

and

$$A\in R^{n\times 2}, \qquad A^T=\begin{bmatrix}1 & 1 & 1 & 1 & \cdots & 1\\ 1 & -1 & 1 & -1 & \cdots & (-1)^n\end{bmatrix},$$

respectively. We tested our methods on the singular versions of all the nonsingular problems except the Gaussian function and the Gulf research and development function, which we excluded because their nonsingular versions never converged to a minimizer using either standard or tensor methods.

Our computational results for the sets of test problems whose Hessians at the minimizers have ranks n, $n-1$, and $n-2$ are summarized in Tables 5.1-5.3, and given in detail in Tables A.1-A.3, respectively. For each problem set, Tables 5.1-5.3 compare the performance of the standard method to two versions of the tensor method: the

TABLE 5.1
Summary of test results for nonsingular problems.

Method	Tensor/std. (itn)	Tensor/std. (fcn)	Tensor better	Std. better	Tie
$p=1$	0.496	0.580	36	5	3
$p\geqq 1$	0.468	0.488	34	5	4

TABLE 5.2
Summary of test results for rank $n-1$ singular problems.

Method	Tensor/std. (itn)	Tensor/std. (fcn)	Tensor better	Std. better	Tie
$p=1$	0.465	0.400	42	3	2
$p\geqq 1$	0.479	0.466	40	5	2

TABLE 5.3
Summary of test results for rank $n-2$ singular problems.

Method	Tensor/std. (itn)	Tensor/std. (fcn)	Tensor better	Std. better	Tie
$p=1$	0.449	0.390	45	1	3
$p\geqq 1$	0.489	0.466	43	5	2

one described in §§ 2–4, where the number of past points interpolated, p, is selected at each iteration to be between 1 and $n^{1/3}$; and a second version, where p is restricted to be 1 at all iterations. We tested the second version because we observed that the first version generally chose $p = 1$ anyhow, and because the tensor method is considerably simpler to implement and is cheaper in terms of storage and cost per iteration, when $p = 1$.

Tables 5.1–5.3 summarize the comparative costs of the standard and tensor methods using ratios of two measures: iterations, and function and derivative evaluations. The iteration ratio is the total number of iterations required by the tensor method on all problems that were successfully solved by both methods, divided by the total number of iterations required by the standard method on these problems. The second ratio is based upon the total number of function evaluations required to solve each problem, including those for finite difference gradients and Hessians (i.e., we count n function evaluations per gradient evaluation and $(n^2+3n)/2$ function evaluations per Hessian evaluation). The ratio reported is the total of these numbers for the tensor method over all problems that were successfully solved by both methods, divided by the total of these numbers for the standard method over the same problems. Tables 5.1–5.3 also contain, for that test set, the number of problems where the performance of the tensor method was better, worse, or the same as the standard method. Here, better is defined as at least 5 percent better in the function evaluation measure; worse is defined as at least 5 percent worse in the function evaluation measure; and the remaining problems are considered the same.

The statistics in Tables 5.1–5.3 only pertain to test problems that were solved successfully by *both* the standard method and the tensor method. Table 5.4 shows, for each test set, how many problems were solved successfully by the tensor method but not by the standard method, and vice versa.

In summary, Tables 5.1–5.4 show that both the $p \geqq 1$ and the $p = 1$ versions of the tensor method have a large advantage, in both reliability and efficiency, over the standard method on all three test sets. In each of the six comparisons, a substantial portion of the test problems (between 16 percent and 22 percent) are solved by the tensor method and not the standard method, while only two problems in the nonsingular sets and none in the singular sets are solved by the standard method and not the tensor method. In addition, on the problems solved by both methods (between 43 and 50 problems in each of the six cases), the average cost of the tensor method, measured in iterations or function and derivative evaluations, is generally slightly less than half of the cost of the standard method. Finally, the improvements by the tensor are quite consistent. Totaling all our tests, the tensor method is worse than the standard method in 8 percent of the test cases (28 of 352), better in 87.5 percent (308 of 352), and the same in 4.5 percent (16 of 352).

The performances of the version of the tensor method that constrains p to be 1, and the version that allows p to be between 1 and $n^{1/3}$ are rather similar overall, with the $p = 1$ version actually performing somewhat better overall on the singular test

TABLE 5.4
Number of problems solved by tensor/standard method only.

Method	Nonsingular	Singular (rank $n-1$)	Singular (rank $n-2$)
$p = 1$	13/2	9/0	13/0
$p \geqq 1$	11/2	12/0	10/0

sets and the $p \geqq 1$ version performing somewhat better on the nonsingular test set. One reason for their similarity is that even when we allow $p > 1$, we have found that our criterion for selecting past iterates to interpolate generally results in $p = 1$. Over all our test problems, we found that the $p \geqq 1$ method selected $p = 1$ 85 percent of the time, $p = 2$ 15 percent, and $p > 2$ 0.35 percent. Thus it appears that the advantages of the tensor method may be achieved by using $p = 1$, which would mean that the extra cost of storing and forming the tensor model would be very low, and that the method would be quite simple to implement. In particular, using $p = 1$ has the advantage that the formulas for T_c and V_c are readily available in closed form in terms of the function and derivative values being interpolated, and that solving the tensor model reduces to minimizing a fourth order polynomial in one variable, which also can be done in closed form.

In our tests, the global portion of the tensor method (steps 5–7 of Algorithm 4.1) selected the step from the quadratic model about 20 percent of the time on the average. While this is a rather small percentage, the performance of the tensor method is improved significantly by allowing this possibility.

We do not claim to fully understand why the tensor method performs so much more efficiently and reliably than a state-of-the-art standard method in our tests. What is especially surprising is that the improvements are attained by incorporating a small amount of extra information, usually just the function in gradient from the previous iterate, into the model. Apparently, having a more accurate model in the direction of the previous step is especially useful in practice.

The computational advantage of the tensor method is probably not due to an improved rate of convergence, except when rank $(\nabla^2 f(x_*)) = n - 1$. In particular, when $\nabla^2 f(x_*)$ is nonsingular and $n > 1$, it is highly unlikely that the convergence rate of the tensor method is different than the quadratic rate of the standard method. (It is easy to show that the tensor method is at least quadratically convergent in this case because the influence of the tensor terms vanishes asymptotically.) In the case when $\nabla^2 f(x_*)$ has rank $n - 1$, we conjecture that the convergence rate of the tensor method is again better than the linear convergence of the standard method, as was shown by Frank [1984] for the tensor method for nonlinear equations. We have not yet attempted to prove this, except in the case $n = 1$, where it is straightforward to show that the tensor method converges with order $(1+\sqrt{7})/3 \approx 1.2$ (Chow [1989]). We did measure the ratios of the errors of successive iterates on our test problems with rank $\nabla^2 f(x_*) = n - 1$. An example is given in Table 5.5. We see that the standard method converges linearly with constant $= \frac{2}{3}$, as predicted by the theory, and that the tensor method appears to be converging faster than linearly. (An interesting feature of this example is that iterations 2 and 5 of the tensor method increase the error in x, even though the function value decreases. We noticed such behavior by the tensor method on several test problems, although for most it did not occur.) When rank $(\nabla^2 f(x_*)) = n - 2$, the tensor method does not have enough information to prove a faster-than-linear convergence rate, since it usually uses $p = 1$.

Finally, we have also implemented a line search version of the tensor method and compared it to an algorithm using the standard quadratic model and the same line search. We found that, on the average, the performances of the line search and trust region versions of the quadratic model algorithms were very similar, and that the line search version of the tensor method was almost 15 percent less efficient than the trust region tensor method. (See Chow [1989] for details.) We observed that the global strategy is only able to use a tensor method step about 60 percent of the time in the line search tensor method, versus about 80 percent of the time in the trust region

TABLE 5.5

Speed of convergence on a typical problem with rank $\nabla^2 f(x_*) = n-1$. *(Singular version of variably dimensioned function,* $n = 10$, *started from* x_0, *using* $p = 1$.) *Numbers in the second and the third columns are* $\|x_k - x^*\| / \|x_{k-1} - x_*\|$.

Iteration (k)	Tensor method	Standard method
1	0.825	0.825
2	61.73	0.825
3	0.028	0.825
4	0.104	0.668
5	7.860	0.776
6	0.033	0.647
7	0.665	0.666
8	0.665	0.665
9	0.600	0.666
10	0.635	0.666
11	0.664	0.666
12	0.654	0.667
13	0.436	0.667
14	0.511	0.667
15	0.120	0.666
16	0.058	0.666
17		0.666
18		0.666
19		0.666
20		0.665
21		0.665
22		0.664
23		0.664
24		0.667

version. This may be related to the difference in their performances. But the line search tensor method still improves by a large amount over the standard method.

6. Summary and future research directions. We have presented a tensor method for unconstrained optimization that bases each iteration upon a fourth order model of the objective function. This model interpolates the function value, gradient, and Hessian of the objective function at the current iterate, and forms its third and fourth order terms by interpolating values of the function and gradient at previous iterates. The costs per iteration of storing, forming, and using the model are not significantly more than for a standard method that uses a quadratic Taylor series model.

The computational results of § 5 show that the tensor method is substantially more reliable and more efficient, in terms of iterations and function and derivative evaluations, than the corresponding standard method on both the nonsingular and singular problems that we tested. This experience indicates that the tensor method may be preferable to methods available in software libraries for solving small- to medium-sized unconstrained optimization problems, in cases when analytic or finite-difference Hessian matrices are available, especially when function and derivative evaluation is the dominant cost. Obviously, more computational experience is necessary to determine this conclusively. To facilitate this process, we are developing a software package that implements a tensor method for unconstrained optimization using analytic or finite-difference second derivatives, and will make it available shortly. Our software package

restricts p, the number of past iterates whose function and gradient values are interpolated at each iteration, to be one. The reasons for this choice are that our computational results show that the tensor method with $p = 1$ is generally about as effective as the method that allows $p \geqq 1$, and that the method is considerably simpler and cheaper to implement in this case. Initially it will use a line search rather than a trust region, because the line search tensor method is currently much easier to understand, and much faster on small, inexpensive problems, than the trust region version, while still leading to large savings in iterations and function and derivative evaluations on our test problems. A trust region version may be added to the package later.

Several interesting research topics remain concerning the tensor method described in this paper. As indicated above, the development of a simple, efficient method for approximately solving the trust region problem using the tensor method would be very useful. Chow [1989] has developed a fairly efficient, but conceptually complicated, method for solving the trust region problem (3.7) when $p = 1$; the question of how to solve this problem efficiently when $p > 1$ remains open and may be an important research issue. It would also be nice to develop an effective global strategy that does not require the determination of the step using *both* the tensor model and the quadratic model at each iteration. It will also be important to test the tensor method on larger-dimensional problems where the linear algebraic costs dominate. The analysis at the end of § 4 indicates that the costs per iteration of the two methods will be very similar if they take the full Newton or tensor steps. In trust region methods, the trust region constraint is often inactive, so this is the case. Otherwise, it will be important to see how many inner iterations per inter iteration the tensor method requires and how this compares to an efficient quadratic trust region method such as Moré and Sorenson [1983]. Finally, as we mentioned in § 5, the local convergence analysis in the case $n > 1$ remains open.

The standard and tensor methods discussed in this paper both assume that the analytic or finite-difference Hessian is available at each iteration. Often in practical applications, however, the analytic Hessian is not available, and it is expensive to calculate by finite differences, so secant (quasi-Newton) methods are used instead. These methods are based on a quadratic model that is formed solely from function and gradient values at the iterates (see, e.g., Dennis and Moré [1977], Fletcher [1980], Dennis and Schnabel [1983]). We are developing a secant tensor method for unconstrained optimization that bases each iteration upon a fourth order model that is also determined solely from function and gradient values at the iterates. This work is described in Chow [1989] and in a forthcoming paper.

Appendix. Test results for the standard and tensor methods. The columns in Tables A.1–A.3 have the following meanings:

—Function: name of the problem.

—n: dimension of the problem.

—x_0: starting point (from Moré, Garbow, and Hillstrom [1981]). 1, 10, and 100 stand for x_0, $10x_0$, and $100x_0$, respectively.

—itns: number of iterations.

—fcns: number of function evaluations (including the necessary function evaluations for finite difference gradients and Hessians).

—x_*: two methods converge to the same minimizer if and only if they have the same letter in this column.

The abbreviations OF, OL, and NC stand for overflow, over iteration limit, and convergence to a nonminimizer, respectively. The iteration limit was 120.

TABLE A.1
Test results for the standard and tensor methods on nonsingular problems.

			Tensor ($p \geqq 1$)			Tensor ($p = 1$)			Standard		
Function	n	x_0	itns	fcns	x^*	itns	fcns	x^*	itns	fcns	x^*
Rosenbrock	2	1	15	150	a	14	146	a	22	183	a
		10	35	345	a	35	345	a	64	542	a
		100	86	833	a	83	795	a	OL	—	—
	10	1	21	1,724	a	21	1,683	a	21	1,615	a
		10	OF	—	—	OF	—	—	73	5,605	a
		100	OF	—	—	OL	—	—	OL	—	—
	30	1	21	11,640	a	25	13,240	a	22	11,610	a
		10	OF	—	—	83	44,067	a	92	48,484	a
		100	OF	—	—	OL	—	—	OL	—	—
Wood	4	1	30	629	a	30	629	a	62	1,230	a
		10	32	674	a	34	723	a	70	1,391	a
		100	OL	—	—	OF	—	—	OL	—	—
Helical valley	3	1	11	170	a	9	138	a	13	175	a
		10	14	213	a	14	215	a	16	216	a
		100	13	194	a	13	194	a	15	203	a
Trigonometric	2	1	4	40	a	4	40	a	4	37	a
		10	6	62	a	6	62	a	7	59	a
		100	4	38	a	4	38	a	5	43	a
	10	1	6	553	a	7	555	a	19	1,467	a
		10	9	708	a	9	708	a	50	3,820	a
		100	40	3,106	a	43	3,336	a	32	2,446	a
Beale	2	1	7	71	a	7	71	a	7	62	a
		10	12	120	a	8	78	a	OL	—	—
		100	OL	—	—	OF	—	—	NC	—	—
Brown and Dennis	4	1	13	272	a	13	271	a	18	347	a
		10	16	333	a	14	292	a	24	461	a
		100	22	469	a	21	440	a	40	778	a
Brown badly scaled	2	1	OL	—	—	OF	—	—	OL	—	—
		10	OL	—	—	OL	—	—	OL	—	—
		100	OL	—	—	OL	—	—	OL	—	—
Box three dimensional	3	1	12	183	a	13	200	a	21	290	a
		10	19	305	a	23	359	a	59	827	b
		100	20	325	a	17	281	b	OL	—	—
Penalty I	4	1	10	211	a	9	190	b	33	644	c
		10	10	209	a	11	229	a	39	757	a
		100	14	293	a	13	272	a	43	829	a
	10	1	15	1,179	a	14	1,101	a	35	2,680	a
		10	11	868	a	9	707	a	40	3,059	a
		100	23	1,811	a	24	1,889	a	52	3,988	a
	30	1	19	10,058	a	20	10,577	a	36	18,973	a
		10	22	11,640	a	26	13,761	a	44	23,189	a
		100	29	15,337	a	31	16,391	a	99	52,174	a
Penalty II	4	1	7	151	a	5	108	a	120	2,285	b
		10	13	274	a	20	416	b	OL	—	—
		100	44	913	a	29	605	b	OL	—	—
Variably dimensioned	4	1	7	153	a	7	153	a	10	195	a
		10	7	150	a	7	148	a	11	214	a
		100	13	272	a	13	271	a	18	348	a

TABLE A.1 (*continued*).

Function	n	x_0	Tensor ($p \geq 1$) itns	fcns	x^*	Tensor ($p = 1$) itns	fcns	x^*	Standard itns	fcns	x^*
	10	1	10	794	a	11	863	a	16	1,229	a
		10	9	936	a	10	786	a	20	1,534	a
		100	18	1,492	a	17	1,338	a	32	2,447	a
	30	1	18	9, 540	a	10	5, 306	a	33	17, 391	a
		10	17	10, 056	a	17	9, 002	a	40	21, 073	a
		100	56	30, 641	a	OL	—	—	112	58, 956	a
Biggs EXP6	6	1	27	972	a	28	1, 005	a	OL	—	—
		10	83	2, 987	a	34	1, 223	b	OL	—	—
		100	25	914	a	26	937	b	OL	—	—
Chebyquad	6	1	6	221	a	6	221	a	14	484	a
		10	34	1, 214	a	29	1, 033	b	OL	—	—
		100	50	1, 785	a	53	1, 885	b	OL	—	—
	20	1	14	3, 815	a	16	4, 064	a	OL	—	—
		10	OF	—	—	95	24, 102	a	OL	—	—
		100	OF	—	—	104	26, 385	a	NC	—	—
Watson	6	1	11	403	a	11	397	a	19	658	a
	20	1	OF	—	—	NC	—	—	OL	—	—

TABLE A.2

Test results for the standard and tensor methods on singular (rank $n-1$) problems.

Function	n	x_0	Tensor ($p \geq 1$) itns	fcns	x^*	Tensor ($p = 1$) itns	fcns	x^*	Standard itns	fcns	x^*
Rosenbrock	2	1	31	305	a	31	305	a	70	570	a
		10	39	383	a	39	383	a	93	764	a
		100	60	601	a	58	579	a	OL	—	—
	10	1	11	945	a	15	1,181	b	15	1,162	c
		10	25	2,049	a	27	2,125	b	27	2,071	c
		100	29	2,272	a	41	3,224	b	66	5,060	c
	30	1	6	3,202	a	6	3,202	a	60	31,598	b
		10	12	6,368	a	12	6,367	b	23	12,137	b
		100	21	11,647	a	21	11,118	a	35	18,453	b
Wood	4	1	25	519	a	25	519	a	37	708	a
		10	34	713	a	36	758	a	52	1,001	a
		100	77	1,576	a	58	1,200	a	OL	—	—
Helical valley	3	1	9	147	a	9	138	a	13	180	a
		10	15	229	a	15	222	a	25	334	a
		100	20	319	a	16	243	a	26	345	a
Trigonometric	2	1	6	59	a	6	59	a	12	102	a
		10	5	50	a	5	50	a	9	75	a
		100	6	58	a	6	58	a	9	76	a

TABLE A.2 (*continued*).

Function	n	x_0	Tensor ($p \geqq 1$) itns	Tensor ($p \geqq 1$) fcns	Tensor ($p \geqq 1$) x^*	Tensor ($p = 1$) itns	Tensor ($p = 1$) fcns	Tensor ($p = 1$) x^*	Standard itns	Standard fcns	Standard x^*
	10	1	8	630	a	7	553	a	14	1,085	a
		10	14	1,097	a	14	1,098	a	15	1,159	a
		100	20	1,639	a	18	1,399	a	NC	—	—
Beale	2	1	6	64	a	7	74	a	6	52	a
		10	10	100	a	10	100	a	47	396	b
		100	64	679	a	23	231	a	44	364	a
Brown and Dennis	4	1	12	254	a	13	274	a	19	366	a
		10	16	337	a	14	293	a	24	461	a
		100	20	421	a	21	443	a	40	778	a
Brown badly scaled	2	1	OF	—	—	OF	—	—	NC	—	—
		10	OL	—	—	OF	—	—	NC	—	—
		100	OL	—	—	OF	—	—	NC	—	—
Box three dimensional	3	1	9	136	a	18	270	b	9	121	c
		10	11	169	a	28	423	b	83	1,083	c
		100	28	430	a	25	380	a	31	412	b
Penalty I	4	1	4	84	a	4	84	a	15	290	a
		10	4	84	a	4	84	a	20	385	a
		100	7	147	a	7	147	a	26	499	a
	10	1	4	318	a	4	318	a	18	1,381	a
		10	7	550	a	7	550	a	24	1,838	a
		100	17	1,341	a	17	1,341	a	35	2,685	a
	30	1	7	3,722	a	10	5,300	a	22	11,605	a
		10	17	9,001	a	16	8,471	a	29	15,292	a
		100	20	10,584	a	18	9,532	a	86	45,334	a
Penalty II	4	1	6	130	a	4	88	a	57	1,098	b
		10	11	229	a	11	229	a	13	252	b
		100	17	354	a	15	313	a	OL	—	—
Variably dimensioned	4	1	4	91	a	4	87	a	17	328	a
		10	9	192	a	11	229	a	19	336	a
		100	17	359	a	18	372	a	25	480	a
	10	1	11	1, 020	a	16	1, 254	a	24	1, 837	a
		10	19	1, 479	a	17	1, 332	a	27	2, 066	a
		100	20	1, 712	a	21	1, 641	a	40	3, 055	a
	30	1	48	25, 382	a	23	12, 170	a	41	21, 599	a
		10	OF	—	—	36	19, 039	a	OL	—	—
		100	OF	—	—	OF	—	—	OL	—	—
Biggs EXP6	6	1	83	2, 962	a	OF	—	—	OL	—	—
		10	74	2, 694	a	63	2, 246	b	OL	—	—
		100	OF	—	—	OF	—	—	OL	—	—
Chebyquad	6	1	9	332	a	10	365	a	92	3, 135	b
		10	22	789	a	19	683	b	OL	—	—
		100	35	1, 245	a	22	785	b	OL	—	—
	20	1	29	7, 349	a	8	2, 045	b	OL	—	—
		10	26	7, 349	a	OF	—	—	OL	—	—
		100	50	13, 903	a	57	14, 441	a	OL	—	—
Watson	6	1	8	295	a	8	295	a	8	297	a
	20	1	24	6, 099	a	28	7, 100	a	OL	—	—

TABLE A.3
Test results for the standard and tensor methods on singular (rank $n-2$) problems.

Function	n	x_0	Tensor ($p \geq 1$) itns	Tensor ($p \geq 1$) fcns	Tensor ($p \geq 1$) x^*	Tensor ($p = 1$) itns	Tensor ($p = 1$) fcns	Tensor ($p = 1$) x^*	Standard itns	Standard fcns	Standard x^*
Rosenbrock	2	1	7	68	a	7	71	a	16	131	b
		10	5	52	a	5	52	a	21	171	b
		100	11	111	a	13	128	b	26	211	c
	10	1	11	872	a	11	872	a	14	1,084	b
		10	33	2,718	a	25	1,959	b	29	2,233	b
		100	25	2,007	a	29	2,330	a	45	3,473	a
	30	1	11	5,841	a	11	5,841	a	21	11,083	a
		10	23	12,172	a	25	1,959	a	OL	—	—
		100	OF	—	—	59	31,247	a	OL	—	—
Wood	4	1	13	277	a	13	275	a	19	366	b
		10	16	339	a	18	378	b	24	461	c
		100	OF	—	—	OF	—	—	NC	—	—
Helical valley	3	1	15	222	a	13	196	a	41	541	a
		10	21	316	a	19	281	a	49	648	a
		100	22	328	a	21	315	a	47	621	a
Trigonometric	2	1	4	38	a	4	38	a	8	67	a
		10	6	62	a	6	62	a	10	83	a
		100	7	70	a	7	70	a	8	67	a
	10	1	6	553	a	7	554	a	15	1,161	a
		10	11	941	a	11	864	a	15	1,159	a
		100	NC	—	—	NC	—	—	NC	—	—
Beale	2	1	6	65	a	6	65	a	10	84	b
		10	10	101	a	9	95	b	10	84	c
		100	22	235	a	20	217	a	NC	—	—
Brown and Dennis	4	1	12	253	a	13	271	a	18	347	a
		10	17	355	a	14	292	a	24	461	a
		100	20	424	a	20	419	a	40	778	a
Brown badly scaled	2	1	NC	—	—	NC	—	—	NC	—	—
		10	NC	—	—	NC	—	—	NC	—	—
		100	NC	—	—	NC	—	—	NC	—	—
Box three dimensional	3	1	16	248	a	11	172	b	16	200	c
		10	22	331	a	13	202	a	22	304	b
		100	48	763	a	40	633	b	46	637	b
Penalty I	4	1	3	64	a	3	64	a	15	290	a
		10	4	84	a	4	84	a	20	385	a
		100	7	147	a	7	147	a	26	499	a
	10	1	4	318	a	4	318	a	18	1,381	a
		10	6	437	a	7	555	a	24	1,838	a
		100	19	1,469	a	18	1,418	a	35	2,685	a
	30	1	9	4,776	a	8	4,246	a	22	11,605	a
		10	17	9,002	a	15	7,944	a	29	15,292	a
		100	16	8,473	a	18	9,525	a	86	45,334	a
Penalty II	4	1	4	88	a	4	88	a	5	100	a
		10	11	230	a	12	253	a	14	271	b
		100	15	314	a	16	335	b	20	385	c
Variably dimensioned	4	1	9	192	a	9	192	a	13	254	b
		10	9	193	a	9	191	a	42	804	b
		100	18	377	a	12	250	b	OL	—	—
	10	1	21	1,663	a	11	868	b	50	3,818	c

TABLE A.3 (*continued*).

Function	n	x_0	Tensor ($p \geqq 1$) itns	Tensor ($p \geqq 1$) fcns	Tensor ($p \geqq 1$) x^*	Tensor ($p = 1$) itns	Tensor ($p = 1$) fcns	Tensor ($p = 1$) x^*	Standard itns	Standard fcns	Standard x^*
		10	16	1,217	a	14	1,100	b	102	7,769	c
		100	24	1,966	a	21	1,666	b	OL	—	—
	30	1	28	15,368	a	16	8,476	b	39	20,547	c
		10	41	23,269	a	OL	—	—	OL	—	—
		100	30	16,426	a	OL	—	—	OL	—	—
Biggs EXP6	6	1	OF	—	—	OF	—	—	OL	—	—
		10	72	2,627	a	54	1,946	b	OL	—	—
		100	41	1,491	a	52	1,884	b	OL	—	—
Chebyquad	6	1	13	473	a	11	404	b	96	3,271	c
		10	30	1,076	a	26	926	a	OL	—	—
		100	37	1,320	a	32	1,141	a	OL	—	—
	20	1	12	3,308	a	13	3,310	a	OL	—	—
		10	OF	—	—	42	10,631	a	OL	—	—
		100	58	15,705	a	46	11,687	a	OL	—	—
Watson	6	1	6	216	a	6	216	a	6	211	a
	20	1	OF	—	—	33	8,369	a	OL	—	—
Powell	4	1	9	194	a	11	233	a	15	290	a
singular		10	12	253	a	14	294	a	21	404	a
		100	22	464	a	22	464	a	26	499	a
	20	1	13	3,308	a	10	2,550	a	16	4,043	a
		10	11	3,308	a	14	3,559	a	21	5,296	a
		100	17	5,075	a	OF	—	—	27	6,801	a

T. CHOW [1989], *Derivative and secant tensor methods for unconstrained optimization*, Ph.D. thesis, Department of Computer Science, University of Colorado, Boulder, CO.

J. E. DENNIS, JR. AND J. J. MORÉ [1977], *Quasi-Newton methods, motivation and theory*, SIAM Rev., 19, pp. 46–89.

J. E. DENNIS, JR. AND R. B. SCHNABEL [1983], *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*, Prentice-Hall, Englewood Cliffs, NJ.

——— [1979], *Least change secant updates for quasi-Newton methods*, SIAM Rev., 21, pp. 443–459.

R. FLETCHER [1980], *Practical Method of Optimization, Vol. 1, Unconstrained Optimization*, John Wiley and Sons, New York.

P. D. FRANK [1984], *Tensor methods for solving systems of nonlinear equations*, Ph.D. thesis, Department of Computer Science, University of Colorado, Boulder, CO.

P. E. GILL, W. MURRAY, AND M. H. WRIGHT [1981], *Practical Optimization*, Academic Press, London.

A. O. GRIEWANK AND M. R. OSBORNE [1983], *Analysis of Newton's method at irregular singularities*, SIAM J. Numer. Anal., 20, pp. 747–773.

R. H. F. JACKSON AND G. P. MCCORMICK [1986], *The polyadic structure of factorable function tensors with appliction to high-order minimization techniques*, J. Optim. Theory Appl., 51, pp. 63–94.

J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM [1981], *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7, pp. 17–41.

J. J. MORÉ AND D. C. SORENSEN [1983], *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4, pp. 553–572.

M. J. D. POWELL [1970], *A new algorithm for unconstrained optimization*, in Nonlinear Programming, J. B. Rosen, O. L. Mangasarian, and K. Ritter, eds., Academic Press, New York, pp. 31–65.

R. B. SCHNABEL AND P. FRANK [1984], *Tensor methods for nonlinear equations*, SIAM J. Numer. Anal., 21, pp. 815–843.

——— [1987], *Solving systems of nonlinear equations by tensor methods*, in The State of the Art in Numerical Analysis, A. Iserles and M. J. D. Powell, eds., Clarendon Press, Oxford, U.K., pp. 245–271.

G. W. STEWART [1970], *Introduction to Matrix Computations*, Academic Press, New York.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 316-332, August 1991

002

THE HOMOTOPY PRINCIPLE AND ALGORITHMS FOR LINEAR PROGRAMMING*

J. L. NAZARETH†

Abstract. Linear programming techniques are formulated within the unifying framework of the homotopy principle and Newton's method. Key strategies that determine the effectiveness of an implementation are considered in detail. A complexity analysis is developed for an elevator predictor, Newton corrector algorithm started at an arbitrary interior primal-dual feasible point. This analysis is based on a fundamental theorem of Smale ["Algorithms for solving equations," *Proc. Internat. Congress of Mathematicians*, University of California, Berkeley, CA, 1986], in the form given by Renegar and Shub [*Report No.* 807, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, 1988].

Key words. linear programming, interior-point methods, homotopy, path-following, prediction-corrector, self-dual, primal-dual, Euler/Newton, complexity analysis

AMS(MOS) subject classifications. 49, 90, 65

1. Introduction. This paper continues work begun in [15]. Within the unifying framework of the homotopy principle and Newton's method (Eaves [5], Smale [21]), we formulate solution techniques for linear programming that subsume piecewise-linear path-following and smooth path-following approaches. The former approach derives from the self-dual simplex method of Dantzig [3] and Lemke [12] and the latter is an offshoot of the interior-point method of Karmarkar [10].

Our paper is organized as follows. In [15] we first showed the relevance of *smooth* path-following homotopy techniques to linear programming, our approach being to specialize certain nonlinear programming results of Garcia and Zangwill [8]; in particular, we defined a homotopy by means of a quadratic regularization of the linear objective function in a suitable metric. We extend this approach in § 2.1 in order to define a homotopy that is now both regular and *boundary free.* We thereby avoid the necessity for restarts used in [15].

An alternative way of defining the homotopy is through the use of a logarithmic (barrier) regularization of the objective function (Frisch [7], Fiacco and McCormick [6]) as studied by Megiddo [13] in a pioneering paper. Barrier functions are a means to an end that can be achieved more easily through *direct parameterization* of the Karush-Kuhn-Tucker optimality conditions as we discuss in § 2.2. Here we are concerned primarily with homotopy paths from *arbitrary* interior starting points. (These too can be achieved by using *weighted* barrier functions (Megiddo [13]) but such barrier functions are again not central to the development.)

As in [15], our approach places us squarely within the path-following framework that is now an accepted way to solve nonlinear equations and *nonlinear* programming problems. We are thus able to draw on this extensive body of knowledge (see, in particular, Garcia and Zangwill [8]) to formulate algorithms within the specialized setting of *linear* programming. We consider, in turn: (a) Elevator and Euler predictor, Newton corrector strategies (§ 3.1). (b) Restarting strategies (§ 3.2). (c) Inexact computation strategies (§ 3.3). As an illustration, in § 3.4, we formulate a particular self-dual

* Received by the editors December 8, 1989; accepted for publication (in revised form) December 10, 1990. This research was supported, in part, by a Washington State University Research and Arts Committee Grant-in-Aid and by National Science Foundation grant DMS-8815513.

† Department of Pure and Applied Mathematics, Washington State University, Pullman, Washington 99164-3113.

algorithm that makes a simple choice for each of these three strategies. (A variety of other algorithms can be formulated, as is also briefly discussed.) Finally, in § 3.5, we combine the preceding ideas with the self-dual simplex approach, returning to the theme that underlies this and our earlier papers [15], [16], namely, the effective integration of simplex and interior-point techniques.

Section 4 deals with the complexity analysis of the approach of § 2.2. We analyse an elevator predictor, Newton corrector algorithm that follows a path from an arbitrary feasible (primal-dual) interior starting point. This analysis is based on a fundamental theorem of Smale [21], our results being patterned after those of Renegar and Shub [19]. Analysis of related weighted barrier algorithms is given by Roos and den Hertog [20] and Tseng [24].

Some concluding remarks are given in the final section.

2. Homotopy techniques. Most discrete iterative procedures defined by a difference equation have an associated continuous version, which generates a trajectory leading from any given starting point to the desired solution and is governed by an underlying differential equation and vector field. By way of introduction, we note here that it is useful to draw a distinction between such *trajectory generating* procedures (a posteriori path definition, for example, Adler, Karmarkar, Resende, and Veiga [1, § 3]) and path-following or homotopy procedures governed by an underlying homotopy differential equation (a priori path definition, for example, Renegar [18]).

2.1. Parameterizing the objective function. Our note [15] was a first attempt at devising an a priori path-following homotopy procedure for linear programming and relating it to a variant of Karmarkar's algorithm [10]. We extend this approach in order to obtain a boundary-free homotopy.

Consider the linear programming problem in standard form and its dual, namely,

$$\begin{array}{lll} \text{(P): minimize } c^T x & & \text{(D): maximize } b^T \pi \\ \text{s.t. } Ax = b & \text{and} & \text{s.t. } A^T \pi + v = c \\ x \geqq 0 & & v \geqq 0, \end{array} \tag{1}$$

where A is an $m \times n$ matrix of full rank, and x, b, and c are vectors of appropriate dimension. We assume throughout that the primal problem (P) is *bounded* and that both (P) and (D) are nondegenerate. Denote the unique optimal solutions of (P) and (D) by x^* and π^*, v^*, respectively.

The Karush–Kuhn–Tucker (KKT) optimality conditions for (1) are as follows.

$$\begin{aligned} Ax &= b \\ A^T \pi + v &= c \\ x_i v_i &= 0, \qquad i = 1, \cdots, n \\ x &\geqq 0, \qquad v \geqq 0. \end{aligned} \tag{2}$$

In a homotopy approach applied to (P), the linear programming problem is parameterized and deformed to a problem that is trivial with a unique solution. Beginning with the solution to the trivial problem, normally a given starting point, a path of solutions is followed as the problem is deformed back to the original linear program. Thus, given a feasible point $x^0 > 0$ for (P), consider the following

parameterized family of programs:

$$\text{minimize}_{x\in R^n}\ \mathscr{F}(x,t)=t(c^Tx)+(1-t)\left\{\frac{1}{2}\sum_{j=1}^{n}\frac{(x_j-x_j^0)^2}{x_jx_j^0}\right\} \tag{3}$$

$$\text{s.t.}\quad Ax=b,\qquad x\geqq 0,$$

where $t\in[0,1]$.

For $i=1,\cdots,n$, the ith component of the gradient of $\mathscr{F}(x,t)$ with respect to x and the ith diagonal element of its (diagonal) Hessian matrix are given by

$$[\nabla_x\mathscr{F}(x,t)]_i\equiv\frac{\partial}{\partial x_i}\mathscr{F}(x,t)=tc_i+\frac{(1-t)}{2}\frac{[(x_i)^2-(x_i^0)^2]}{(x_i)^2(x_i^0)} \tag{4}$$

$$[\nabla_x^2\mathscr{F}(x,t)]_{ii}=(1-t)\frac{x_i^0}{(x_i)^3}. \tag{5}$$

The foregoing Hessian matrix has *positive* diagonal elements on R_+^n, the positive orthant. This would not be true if the denominator in the quadratic regularizing term in (3) were to be replaced, more simply, by $(x_j)^2$. The Hessian matrix could then become singular. Other choices for this denominator are possible; for example, we initially considered $\min[(x_j)^2,(x_j^0)^2]$, but the choice $x_jx_j^0$, suggested by Karmarkar, may be preferable.

When $t=0$, the unique solution of (3) is $x[0]\equiv x^0>0$. When $t=1$, the solution of (3) is $x[1]\equiv x^*$, the unique optimal solution of (P). For $t\in[0,1)$, the objective function of (3) is strictly convex on R_+^n. Therefore, for each $t\in[0,1)$, the program (3) has a unique optimal solution in R_+^n, say, $x[t]>0$. The points $x[t]$ trace a path of solutions from the given starting point x^0 and the homotopy that we have defined is clearly *boundary free* for $t\in[0,1)$ (terminology of Garcia and Zangwill [8]).

A solution $x[t]$, $t\in[0,1)$, is obtained from the following KKT system:

$$\begin{aligned}-\nabla_x\mathscr{F}(x,t)+A^T\pi&=0,\\ Ax&=b,\end{aligned} \tag{6}$$

where $\pi\equiv\pi[t]$ is the vector of Lagrange multipliers.

The path of solutions is governed by a *homotopy differential equation* (HDE). Define $z^T\equiv(x,\pi)^T$, and denote the system of equations (6) by

$$H(z[t],t)=0.$$

Then the HDE is defined as follows:

$$\frac{dz}{dt}=-[H_z']^{-1}[H_t'], \tag{7}$$

where

$$H_z'=\begin{bmatrix}-\nabla_x^2\mathscr{F}(x,t) & A^T\\ A & 0\end{bmatrix},\qquad H_t'=\begin{bmatrix}-(\partial/\partial t)\,\nabla_x\mathscr{F}(x,t)\\ 0\end{bmatrix}. \tag{8}$$

The assumption that A is of full rank together with (5) then implies that H_z' is *nonsingular* over $x\in R_+^n$, $t\in[0,1)$, i.e., the homotopy is also *regular*. Path existence follows from results given by Garcia and Zangwill [8] that are based, in turn, on the implicit function theorem. (Path boundedness, implying that it reaches x^* when $t=1$, follows from direct modification of results given in [8, § 4.5] for a related problem.)

The path can be followed using techniques analogous to those discussed in § 3 of this paper.

We note that an initial linear approximation (Euler step) at $x[0]$ to the path defined by (7) and (8) will give an affine (*scaling*) *direction of descent* (Dikin [4]). The derivation is analogous to that used to obtain expression (3.13) in [15]. This direction $d[0]$ is as follows:

$$d[0] = -D_0[I - D_0A^T(AD_0^2A^T)^{-1}AD_0]D_0c, \tag{9}$$

where $D_0 = \operatorname{diag}[x_1^0, \cdots, x_n^0] > 0$. However, at an arbitrary point on the path, $t \in (0, 1)$, an Euler step will not usually give an affine (scaling) direction.

2.2. Parameterizing the KKT conditions directly. We derive a homotopy directly from the KKT conditions (2). Given $x^0 > 0$ feasible for (P) and π^0, v^0 feasible for (D) with $v^0 > 0$ (the feasibility assumption is removed in § 3.5), we parameterize the KKT conditions (2) in the most immediate way that presents itself, namely,

$$\begin{aligned} Ax &= b \\ A^T\pi + v &= c \\ x_iv_i &= (1-t)x_i^0v_i^0, \qquad i = 1, \cdots, n \\ x \geqq 0, &\qquad v \geqq 0, \end{aligned} \tag{10}$$

where $t \in [0, 1]$. When $t = 0$, the point x^0, π^0, v^0 is a solution of (10). When $t = 1$, then (10) is identical to (2) and solves (P) and (D). Let us write the first three equations of this system more compactly as $\mathcal{H}(x, \pi, v, t) = 0$. In order to establish that the homotopy defined by (10) is regular and the path is bounded and leads from x^0, π^0, v^0 ($t = 0$) to x^*, π^*, v^* ($t = 1$), we proceed via a reduction to a theorem of Dikin [4] as follows.

Interpret (10) as the KKT conditions that correspond to the parameterized problem[1] $\min [c^Tx - (1-t)\sum_i (x_i^0v_i^0) \ln (x_i)]$ such that $Ax = b$, $x \geqq 0$, and observe that for given $t \in [0, 1)$ the objective function is strictly convex, and the optimal solution $x[t]$ lies in R_+^n. Here we view the introduction of the weighted logarithmic barrier function as a convenient means to establishing properties of the system (10) and not as central to the development. An alternative proof that $x[t] > 0$ would do just as well. With the restriction $x \in R_+^n$, we have

$$v_i = (1-t)(x_i^0v_i^0)/x_i, \qquad i = 1, \cdots, n. \tag{11}$$

Since $x[t] > 0$, expression (11) implies that the corresponding $v[t] > 0$, $t \in [0, 1)$.

We now establish that $\pi[t]$, $v[t]$ are bounded for $t \in [0, 1)$. The first two equations of (10) can be re-expressed as follows:

$$\begin{aligned} Ax &= b, \\ A^T\pi + (1-t)D_{v,x}x^0 &= c, \end{aligned} \tag{12}$$

where $D_{v,x} \equiv \operatorname{diag}[(v_1^0/x_1), \cdots, (v_n^0/x_n)] > 0$. Thus we deduce that

$$(1-t)Ax^0 + (AD_{v,x}^{-1}A^T)\pi = AD_{v,x}^{-1}c.$$

The expression $(AD_{v,x}^{-1}A^T)$ is invertible because A is of full rank and $D_{v,x} > 0$. Also, $Ax^0 = Ax$ $(= b)$ implies that $Ax^0 = AD_{v,x}^{-1}v^0$. Thus

$$\pi = (AD_{v,x}^{-1}A^T)^{-1}AD_{v,x}^{-1}c - (1-t)(AD_{v,x}^{-1}A^T)^{-1}AD_{v,x}^{-1}v^0. \tag{13}$$

[1] It is worth noting that use of the convex combination $[tc^Tx - (1-t)\sum_i (x_i^0v_i^0) \ln (x_i)]$ would give an alternative parameterization of the KKT conditions with c replaced by (tc) in (10). This corresponds to starting *at* the weighted center of the primal polytope.

It follows directly from a theorem of Dikin (see Vanderbei and Lagarias [25] and Stewart [23]) that $\pi[t]$ is bounded for any $t \in [0, 1)$. Hence $v[t] > 0$ is bounded. Recall also that the primal problem is assumed to be bounded and nondegenerate.

Given regularity of the homotopy (the Jacobian matrix of (12) is of full rank) and $x[t]$, $\pi[t]$, $v[t]$ bounded, we can then deduce, following the argument given in Garcia and Zangwill [8], that the homotopy defined by the system (12) does indeed generate a path leading from x^0, π^0, v^0 to x^*, π^*, v^*, the optimal solution of (P) and (D).

Denote the system (12) by

$$H(z[t], t) = 0, \tag{14}$$

where $z^T \equiv (x, \pi)^T$. The associated homotopy differential equation is given by

$$\frac{dz}{dt} = -[H'_z]^{-1}[H'_t], \tag{15}$$

where

$$H'_z = \begin{bmatrix} -(1-t)D_{x,v}^{-2} & A^T \\ A & 0 \end{bmatrix}, \qquad H'_t = \begin{bmatrix} w_{x,v} \\ 0 \end{bmatrix}, \tag{16}$$

and

$$\begin{aligned} D_{x,v} &\equiv \operatorname{diag}[x_1/\sqrt{(x_1^0 v_1^0)}, \cdots, x_n/\sqrt{(x_n^0 v_n^0)}], \\ w_{x,v}^T &\equiv -[(x_1^0 v_1^0)/x_1, \cdots, (x_n^0 v_n^0)/x_n]. \end{aligned} \tag{17}$$

We observe that H'_z is nonsingular when $x \in R_+^n$, $t \in [0, 1)$. In § 3 we shall consider strategies for following paths defined by (15)-(17).

Impose the further assumption, essentially combinatorial in nature, that the initial x^0, π^0, v^0 lies on the so-called central path (Sonnevend [22], Bayer and Lagarias [2], Megiddo [13]), namely,

$$x_i^0 v_i^0 = \theta, \qquad i = 1, \cdots, n, \tag{18}$$

where θ is some given positive number. (Note that $\theta = 0$ enforces complementary slackness.) Then the HDE (15)-(17) becomes

$$\frac{dz}{dt} = -[H'_z]^{-1}[H'_t], \tag{19}$$

where

$$H'_z = \begin{bmatrix} -(1-t)D_x^{-2} & A^T \\ A & 0 \end{bmatrix}, \qquad H'_t = \begin{bmatrix} -D_x^{-1}e \\ 0 \end{bmatrix}, \tag{20}$$

and $D_x \cong \operatorname{diag}[x_1, \cdots, x_n]$, $e^T \equiv (1, 1, \cdots, 1)$. We use the symbol $\cong$ to indicate that quantities on either side of the symbol are a scalar multiple of one another.

An Euler step at any point on the path defined by (15) is then obtained by solving

$$\begin{bmatrix} -(1-t)D_x^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \pi \end{bmatrix} = \begin{bmatrix} D_x^{-1}e \\ 0 \end{bmatrix}. \tag{21}$$

Simple manipulations then establish the following:

$$\Delta x \cong -[I - D_x^2 A^T (AD_x^2 A^T)^{-1} A] D_x e = -D_x [I - D_x A^T (AD_x^2 A^T)^{-1} A D_x] D_x c \tag{22}$$

$$\begin{bmatrix} \Delta \pi \\ \Delta v \end{bmatrix} \cong \begin{bmatrix} I \\ -A^T \end{bmatrix} (AD_v^{-2} A^T)^{-1} b, \tag{23}$$

where $D_v = \operatorname{diag}[v_1, \cdots, v_n] \cong D_x^{-1}$.

These are the usual primal and dual affine (scaling) directions used simultaneously, and with sufficiently small steps they will therefore trace the central path to any desired degree of accuracy ([8, Thm. 16.2.1]).

3. Algorithms and solution procedures. In this section we consider procedures for following paths from an *arbitrary* feasible starting point x^0, π^0, v^0 for the system (15)-(17). We consider, in turn,

(a) The Euler predictor/Newton corrector strategy.
(b) The restarting strategy.
(c) The inexact computation strategy.

These three strategies hold the key to effective implementation, and a variety of algorithms can be derived by making particular choices for them. For background on items (a) and (b), see Garcia and Zangwill [8].

3.1. The Euler/Newton strategy for following a path. Most of the theoretical studies to date, for instance, [18], [19], utilize elevator (vertical) predictor steps and Newton corrector steps. See [8] for terminology. However, use of Euler predictor steps (cf. [15]) is likely to be more effective in practice. The derivation of an Euler step for (15)-(17) is analogous to the derivation of (22)-(23) from (19)-(20). Thus, linearizing at any point $z^T = (x, \pi)^T$ on the path, we obtain the Euler step $\Delta z^T = (\Delta x, \Delta \pi)^T$ as the solution of the following system of equations.

$$\begin{bmatrix} -(1-t)D_{x,v}^{-2} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \pi \end{bmatrix} = - \begin{bmatrix} w_{x,v} \\ 0 \end{bmatrix} \tag{24}$$

where $D_{x,v}^{-2}$ and $w_{x,v}$ are defined in (17).

Straightforward manipulation and use of the relations $-AD_{x,v}^2 w_{x,v} = Ax = b$ then gives

$$\Delta \pi \cong (AD_{x,v}^2 A^T)^{-1} b, \qquad \Delta v \cong -A^T (AD_{x,v}^2 A^T)^{-1} b, \tag{25}$$

$$\Delta x \cong -D_{x,v}[I - D_{x,v}A^T(AD_{x,v}^2 A^T)^{-1} AD_{x,v}](-D_{x,v} w_{x,v}). \tag{26}$$

Denote the matrix in square brackets in expression (26) by $P_{x,v}$. Then the relations $(-D_{x,v} w_{x,v}) = [\sqrt{x_1^0 v_1^0}, \cdots, \sqrt{x_n^0 v_n^0}]^T \cong D_{x,v} v$, $v = c - A^T \pi$ and $D_{x,v} P_{x,v} D_{x,v} A^T \pi = 0$ enable us to re-express (26) as follows:

$$\Delta x \cong -D_{x,v}[I - D_{x,v}A^T(AD_{x,v}^2 A^T)^{-1} AD_{x,v}] D_{x,v} c. \tag{27}$$

Also, (25) can be stated as follows:

$$\begin{bmatrix} \Delta \pi \\ \Delta v \end{bmatrix} \cong \begin{bmatrix} I \\ -A^T \end{bmatrix} (AD_{x,v}^2 A^T)^{-1} b. \tag{28}$$

As before, the symbol $\cong$ means that quantities on either side of the symbol are scalar multiples of one another. Note also that we are dealing with a point (x, π, v) that lies on the path, so $D_{x,v}$ can be written in alternative ways, for example, $D_{x,v} \cong \operatorname{diag}[\sqrt{x_1/v_1}, \cdots, \sqrt{x_n/v_n}]$. This will not be true when iterates stray off the path.

The elevator and Euler predictor steps are depicted in Fig. 1. Next consider the corrector step at the point $(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k)$ depicted there. (The expressions defining the Newton corrector are very similar to the ones given above for the Euler predictor and are not detailed here.) From (27) and (28) it is clear that $\tilde{x}^k$ is feasible for (P) and $\tilde{\pi}^k, \tilde{v}^k$ are feasible for (D). However, in general, $\tilde{x}_i^k \tilde{v}_i^k \neq (1-t^k) x_i^0 v_i^0$, i.e., the third equation of (10) will not be satisfied. Keeping t fixed at t^k, Newton's method can be applied to the system (10) starting at the point $(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k)$, with iterates for the variables

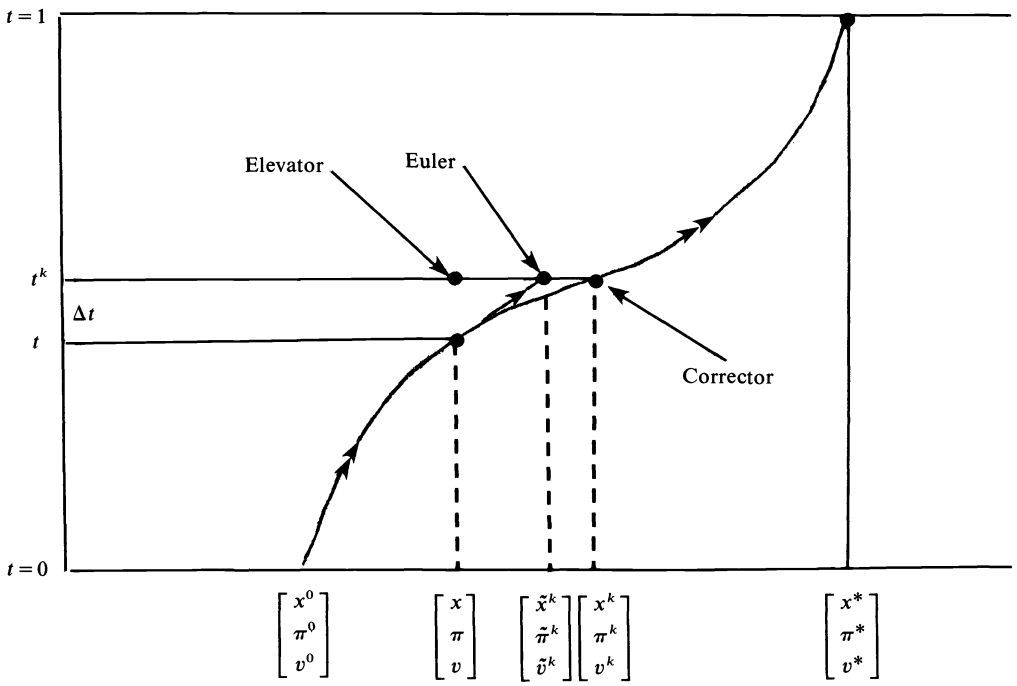


FIG. 1. *Predictor/corrector steps.*

x and v confined to R^n_+. Convergence of these iterates to the point on the path corresponding to $t = t^k$ taken to be (x^k, π^k, v^k) in Fig. 1 can be assured by taking Δt sufficiently small so that $(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k)$ is sufficiently close to the path; indeed, it follows from Theorem 16.2.1 of [8] that the path can be followed to any degree of accuracy by the Euler (or elevator) predictor and Newton corrector strategy. (In practice, of course, one would seek to take much larger predictor steps.)

Keeping t fixed corresponds to the so-called *horizontal* corrector. It is also useful to note that t can be treated as a variable during the corrector phase, and a further linear constraint can be imposed in order to ensure, for example, that the objective function values of the primal (dual) do not increase (decrease). For a good discussion, see [8].

3.2. Restarting the homotopy. Since we are dealing here with a *terminal value problem*, restarts are useful, as is well motivated by Watson [26] as follows:

> This means that the curve itself is not important, and sophisticated homotopy algorithms (as in HOMPACK, for example) actually *change* the curve that is being tracked as they proceed. This strategy has a rigorous theoretical justification, since changing the curve amounts to changing the parameter vector in the homotopy map, and for almost all parameter vectors the zero curve of the homotopy map is guaranteed to reach a solution. However, following the zero curve too loosely can be disastrous, so there is a delicate balance between efficiency and robustness. This balance needs further study, perhaps leading to fundamentally different algorithms.

Consider again the iterate $(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k)$, which has strayed from the path in Fig. 1, and *redefine* the homotopy equations as follows:

$$\mathcal{G}(x, \pi, v, t) = \mathcal{H}(x, \pi, v, t) - \frac{(1-t)}{(1-t^k)}\,\mathcal{H}(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k, t^k), \tag{29}$$

where $\mathcal{H}(\cdot)$ denotes the system (10). When $t = t^k$, then $\mathcal{G}(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k, t^k) = 0$. When

$t=1$, then $\mathcal{G}(x, \pi, v, 1)=\mathcal{H}(x, \pi, v, 1)$. Hence $\mathcal{G}(x^*, \pi^*, v^*, 1)=0$. Therefore the *restarted* homotopy defines a path from $(\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k, t^k)$ to the unique optimal solution.

Let us consider the system $\mathcal{G}(x, \pi, v, t)$ explicitly. Because $\tilde{x}^k, \tilde{\pi}^k, \tilde{v}^k$ satisfy the first two equations of (10) exactly, the first two equations of $\mathcal{G}(\cdot)$ and $\mathcal{H}(\cdot)$ are identical. (The Newton corrector would also satisfy these equations exactly.) Consider, therefore, the third set of equations. From (29),

$$\mathcal{G}(x, \pi, v, t)=x_j v_j-(1-t)x_j^0 v_j^0-\frac{(1-t)}{(1-t^k)}[\tilde{x}_j^k \tilde{v}_j^k-(1-t^k)x_j^0 v_j^0]=0. \tag{30}$$

Hence

$$x_j v_j=\frac{(1-t)}{(1-t^k)}\tilde{x}_j^k \tilde{v}_j^k. \tag{31}$$

The restarted homotopy is thus defined, again in a natural way, by

$$\begin{aligned} Ax &= b, \\ A^T\pi + v &= c, \\ x_j v_j &= \frac{(1-t)}{(1-t^k)}\tilde{x}_j^k \tilde{v}_j^k, \qquad j=1, \cdots, n, \\ x &\geqq 0, \qquad v \geqq 0, \end{aligned} \tag{32}$$

where $t \in [t^k, 1]$.

3.3. Inexact computation. Suppose Z spans the null space of A and is of full rank. For example,

$$Z \equiv \begin{bmatrix} -B^{-1}N \\ I_{\bar{n}\times\bar{n}} \end{bmatrix},$$

where $A=[B \mid N]$, B is a nonsingular basis matrix, and N is the matrix whose columns correspond to the remaining $\bar{n}=n-m$ nonbasic variables. Note that these nonbasic variables are not at their bounds. Then the range-space expression (27) can be stated in the alternative *null-space* form

$$\Delta x \cong -Z(Z^T D_{x,v}^{-2} Z)^{-1} Z^T c. \tag{33}$$

In expression (28), the matrix

$$\begin{bmatrix} I \\ -A^T \end{bmatrix}$$

spans the null space of the dual constraints $[A^T \mid I]\left[\begin{smallmatrix}\pi\\ v\end{smallmatrix}\right]=c$, and $\left[\begin{smallmatrix}\Delta x\\ \Delta\pi\end{smallmatrix}\right]$ is already in the null-space form. Indeed, the corresponding range-space form does not exist for the dual (D). In contrast, the symmetric primal-dual statement of the linear programs will have range-space and null-space forms for both the primal and dual directions.

The advantage of the null-space forms is that the corresponding directions can be computed inexactly, for example, by a few iterations of the conjugate gradient algorithm, without loss of feasibility. Moreover, these inexactly computed directions continue to be directions of descent (ascent) for the primal (dual). This is discussed in detail in [16] and similar considerations apply to (33) and (28). (In particular, use of iterative methods like conjugate gradients is essential when computing with null-space formulations because we do not want to form the matrix $B^{-1}N$ explicitly. Note that $B^{-1}N$ and $(Z^T D_{x,v}^{-2} Z)$ are, in general, dense matrices.)

3.4. Resulting algorithms. In order to provide an illustration, we consider a simple choice for each of the preceding strategies, namely, (a) Euler predictor steps, (b) a

restart after each step, and (c) inexact computation of each Euler step using an iterative method, for example, a few iterations of the conjugate gradient method. The resulting algorithm is as follows:

ALGORITHM SELF-DUAL AFFINE (SCALING):
1. $k := 0$, $\tilde{x}^0 := x^0$, $\tilde{\pi}^0 := \pi^0$, $\tilde{v}^0 := v^0$.
2. $D_{x,v} := \text{diag}\,[\sqrt{\tilde{x}_1^k/\tilde{v}_1^k}, \cdots, \sqrt{\tilde{x}_n^k/\tilde{v}_n^k}]$.
3(a). Solve $(Z^T D_{x,v}^{-2} Z) d_x = -Z^T c$ approximately by an iterative method to obtain a direction d_x.
3(b). $\tilde{x}^{k+1} := \tilde{x}^k + \gamma Z d_x$, where Zd_x is a direction of descent for the primal problem and γ is a suitable step size that maintains primal feasibility.
4. Solve $(AD_{x,v}^2 A^T) d_\pi = b$ approximately by an iterative method to obtain a direction of ascent d_π for the dual problem.
5. $d_v := -A^T d_\pi$.
6(a). $\tilde{v}^{k+1} := \tilde{v}^k + \beta d_v$, where β is a suitable step size that maintains dual feasibility.
6(b). $\tilde{\pi}^{k+1} := \tilde{\pi}^k + \beta d_\pi$.
7. If $(c^T \tilde{x}^{k+1} - b^T \tilde{\pi}^{k+1}) < eps$ then stop, else $k := k+1$ and return to Step 2.

When search directions are computed exactly and $\beta = \gamma (\equiv \alpha)$ are chosen appropriately, one obtains the algorithm analysed by Monteiro, Adler, and Resende [14]. A variety of other algorithms can be formulated within the framework of §§ 3.1–3.3. For example, by using elevator predictor and Newton corrector steps, exact computation of search directions, and no restarts, one obtains the algorithm of Kojima, Mizuno, and Yoshise [11], which has roots in Megiddo [13]. Again, by using a pure Euler strategy (i.e., without corrector steps), exact computation of search directions, and no restarts, one obtains a new algorithm that simultaneously computes primal and dual affine (scaling) directions from the scaling matrix $D_x = \text{diag}\,[\tilde{x}_1^k/\sqrt{x_1^0 v_1^0}, \cdots, \tilde{x}_n^k/\sqrt{x_n^0 v_n^0}]$. Note that as iterates stray off the path, D_x will no longer be a scalar multiple of the matrix $\text{diag}\,[\sqrt{\tilde{x}_1^k/\tilde{v}_1^k}, \cdots, \sqrt{\tilde{x}_n^k/\tilde{v}_n^k}]$. Use of the latter scaling matrix would thus give a different algorithm.

For each such algorithm, the Garcia–Zangwill framework (in particular, Theorem 16.2.1) demonstrates that the homotopy path can be followed to any desired degree of accuracy. Recently, Renegar and Shub [19] showed how the complexity analysis of several elevator predictor/Newton corrector algorithms can be greatly simplified using a fundamental theorem of Smale [21]. Their approach to complexity analysis applies to the study of algorithms formulated within the framework described here, as will be discussed in § 4.

We should also note that completely analogous considerations to those discussed in §§ 3.1–3.3 apply to the system (7)–(8). These too are currently under study and will be reported elsewhere.

3.5. Combining with the self-dual simplex algorithm. The self-dual (parametric) simplex algorithm of Dantzig [3], a special case of Lemke's algorithm [12] for the linear complementarity problem, parameterizes the KKT system (2) as follows:

$$\begin{aligned} Ax &= b + (1-t)q_p \\ A^T\pi + v &= c + (1-t)q_d \\ x_i v_i &= 0, \qquad i = 1, \cdots, n \\ x \geqq 0, &\qquad v \geqq 0, \end{aligned} \tag{34}$$

where $t \in [0, 1]$. The conditions (34) are the optimality conditions for associated parameterized versions of the linear programs (P) and (D). For any given basis matrix of (P), the vectors q_p and q_d are chosen so that the corresponding basic solutions of these *parameterized* programs are feasible. Obviously, these solutions satisfy the complementary slackness conditions and thus solve (34). (Note that they need not be feasible for the original unparameterized linear programs.) A piecewise-linear path of solutions to (34) is followed as t varies from 0 to 1.

When primal and dual feasible solutions are not available, as assumed in § 2.2, we utilize the foregoing approach to parameterize the KKT system (2) as follows:

$$\begin{aligned} Ax &= b + (1-t)q_p \\ A^T\pi + v &= c + (1-t)q_d \\ x_i v_i &= (1-t)(x_i^0 v_i^0), \qquad i = 1, \cdots, n \\ x \geqq 0, &\qquad v \geqq 0. \end{aligned} \tag{35}$$

Here x^0 and v^0 are arbitrary n-vectors with *positive* components, and π^0 is an arbitrary m-vector. The vectors q_p and q_d are given by $q_p = (Ax^0 - b)$, $q_d = (A^T\pi^0 + v^0 - c)$. The system (35) reduces to the self-dual simplex system (34) when x^0 and v^0 satisfy the complementary slackness conditions. Therefore, note especially that (35) subsumes the smooth homotopy and piecewise-linear homotopy systems given by (10) and (34), in a natural manner.

A development completely analogous to § 2.2 can be carried out, and path-following algorithms derived, as discussed in § 3. Then the expressions analogous to (15)–(16) are as follows.

$$\frac{dz}{dt} = -[H'_z]^{-1}[H'_t], \tag{36}$$

where

$$H'_z = \begin{bmatrix} -(1-t)D_{x,v}^{-2} & A^T \\ A & 0 \end{bmatrix}, \qquad H'_t = \begin{bmatrix} w_{x,v} + q_d \\ q_p \end{bmatrix}. \tag{37}$$

The fact that a homotopy path for the foregoing system can be followed to any degree of accuracy by an algorithm based on an elevator or Euler predictor together with a Newton corrector again follows from Theorem 16.2.1 of [8]. A complexity analysis can be based on Smale [21] and Renegar and Shub [19] along lines similar to § 4 of our paper. These topics are under investigation and will be reported elsewhere.

Note also that the self-dual simplex algorithm provides a convenient way to obtain an optimal vertex from the point where a smooth path-following algorithm terminates. For example, the self-dual simplex algorithm can be initiated from a basis given by the m variables farthest from their bounds, with the remaining (nonbasic) variables set to their zero bounds. The only requirement is that the basis matrix be nonsingular; the associated basic solutions need not be feasible for primal or dual.

In the next section we turn to the analysis of a particular homotopy algorithm.

4. Complexity analysis. In § 2.2 we established the existence of a path leading from (x^0, π^0, v^0) to the optimal solution for the homotopy system derived from (10). Theorem 16.2.1 of Garcia and Zangwill [8] then establishes that by taking sufficiently small steps this path can be followed to *an arbitrary degree of accuracy* using an elevator predictor, Newton corrector algorithm. Now we turn to a complexity analysis of such an algorithm based on a theorem of Smale [21], in the form given by Renegar and

Shub [19]. (In the interests of brevity, we give a condensed version of our proofs. Full details can be found in [17].)

Let $\mathscr{X}$ and $\mathscr{Y}$ denote Banach spaces, where the norm on $\mathscr{X}$ is $\|\ \|$. For an operator $M: \mathscr{X}^k \to \mathscr{X}$, let $\|M\|$ denote the usual operator norm $\|M\| = \sup\{\|M(u^{(1)}, \cdots, u^{(k)})\|; \|u^{(i)}\| = 1, \forall i\}$. Assume that $f: \mathscr{X} \to \mathscr{Y}$ is an analytic map. Let Df denote the Frechet derivative. If Df^{-1} exists at x, define

$$\beta(x) = \|Df(x)^{-1} f(x)\|, \tag{38}$$

$$\gamma(x) = \sup_{k \geqq 2} \left\| \frac{1}{k!} Df(x)^{-1} D^k f(x) \right\|^{1/(k-1)}. \tag{39}$$

THEOREM (Smale [21], in the formulation of Renegar and Shub [19]). *Assume that $f: \mathscr{U} \to \mathscr{Y}$ is analytic, where $\mathscr{U}$ is an open subset of $\mathscr{X}$. Assume that $\beta = \beta(\xi)$, $\gamma = \gamma(\xi)$ satisfy $\beta \leqq \frac{1}{2}\delta \leqq 1/40\gamma$ and $B(\xi, 4\delta) \subseteq \mathscr{U}$. If $\|\bar{\xi} - \xi\| \leqq \delta$, then the Newton sequence $x^{(0)} = \bar{\xi}$, $x^{(i+1)} = x^{(i)} - Df(x^{(i)})^{-1} f(x^{(i)})$ is well defined and converges to the zero ξ' of f in $B(\xi, \frac{3}{2}\beta)$, this being the only zero of f in $B(\xi, 4\delta)$. Moreover,*

$$\|x^{(i)} - \xi'\| \leqq \tfrac{7}{2}(\tfrac{1}{2})^{2^i}\delta. \tag{40}$$

□

For any vector $d \in R^n$, let $\|d\|_2$ denote the Euclidean norm. Let Ω be any diagonal matrix, and let $\|\Omega\|_2$ denote the matrix norm that is subordinate to the Euclidean vector norm. Let $\|d\|_{W^{-1}} = \sqrt{d^T W^{-1} d}$, where $W > 0$ is a given diagonal matrix with positive diagonal elements. Let $\|\Omega\|_{W^{-1}}$ denote the norm of Ω subordinate to the vector norm just defined.

We shall utilize the following results concerning these diagonal matrices and associated norms. Their proofs are simple and are therefore omitted.

PROPOSITION 1.

$$\|\Omega\|_2 = \max \Omega_{ii} \leqq \|\Omega e\|_2 = \sqrt{\Omega_{11}^2 + \cdots + \Omega_{nn}^2}.$$

PROPOSITION 2.

$$\|\Omega\|_{W^{-1}} = \|\Omega\|_2.$$

When the homotopy system derived from the first three equations of (10) is expressed in terms of the variables x and π (with v being defined by the second equation and eliminated from the third), we obtain the following:

$$X\Delta(\pi)e - \mu We = 0, \qquad b - Ax = 0, \tag{41}$$

where $X = \operatorname{diag}[x_1, \cdots, x_n]$, $\Delta(\pi) = \operatorname{diag}[c_1 - a_1^T\pi, \cdots, c_n - a_n^T\pi]$, with a_j being the jth column of A. Also, $W = \operatorname{diag}[x_1^0 v_1^0, \cdots, x_n^0 v_n^0] > 0$, $e = (1, 1, \cdots, 1)$, and $\mu = (1 - t)$. Let us assume that W has bounded spectral condition number, namely,

$$\kappa_2(W) = \|W\|_2 \|W^{-1}\|_2 = \max_j [(c^T - a_j^T\pi^0)x_j^0] / \min_j [(c^T - a_j^T\pi^0)x_j^0]. \tag{42}$$

Let $w = (\pi, x) \in R^{m+n}$ and let $H(w, \mu) = 0$ denote the system (41). The specific algorithm that we consider here for following the path defined by $H(w, \mu) = 0$ is an elevator predictor, Newton corrector algorithm and it is defined by

$$w^{(i+1)} = w^{(i)} - D_w H(w^{(i)}, \mu^{(i+1)})^{-1} H(w^{(i)}, \mu^{(i+1)}), \tag{43}$$

where $\mu^{(i)} \downarrow 0$. D_w denotes the Frechet derivative with respect to w.

We now claim that we can take

$$\mu^{(i+1)} = \left(1 - \frac{1}{40\sqrt{n}\sqrt{\|W\|_2 \|W^{-1}\|_2}}\right)\mu^{(i)}, \tag{44}$$

and that each $w^{(i)}$ will be a good approximation to the zero of $w \to H(w, \mu^{(i)})$, with π and x, the components of w, being interior points of dual and primal, respectively. Our proof is patterned after § 5 of Renegar and Shub [19], the difference between their results and ours being that the *central* path-following algorithm is analysed in [19], whereas we study the path from an *arbitrary* interior feasible point (π^0, x^0).

Fix $\hat{w} = (0, \hat{x})$ satisfying $A\hat{x} = b$. Let $\mathscr{X} = \{(\pi, x) \in R^{m+n}, Ax = 0\}$ and let the components of $w^{(0)}$ be feasible for dual and primal, respectively. Subsequently, we shall maintain $w^{(i)} - \hat{w} \in \mathscr{X}$ for all i, and $w = (\pi, x)$ will always refer to points in $\mathscr{X}$.

Assume $\mu > 0$ fixed. Let $\xi = (\rho, \omega) \in \mathscr{X}$ be the unique zero of $w \to H(w + \hat{w}, \mu)$ where ρ is an interior point for the dual. Let $\Omega = \operatorname{diag}[\omega_1, \cdots, \omega_n]$ and $\hat{X} = \operatorname{diag}[\hat{x}_1, \cdots, \hat{x}_n]$. Then

$$(\Omega + \hat{X})\Delta(\rho)e - \mu We = 0. \tag{45}$$

Hence $(1/\mu)W^{-1}(\Omega + \hat{X})\Delta(\rho)e = e$ and diagonality of all the matrices in this expression imply that

$$\frac{1}{\mu} W^{-1}(\Omega + \hat{X})\Delta(\rho) = I. \tag{46}$$

Now we are able, in fact, to choose a starting point *on* the path itself. Therefore we certainly have no difficulty in choosing $\bar{\xi} = (\bar{\rho}, \bar{\omega})$ initially to satisfy

$$\|\bar{\xi} - \xi\| \leqq \delta \quad \text{with} \quad \delta = \frac{1}{20\| W^{-1/2}\|_2} \tag{47}$$

for an appropriate choice of vector norm $\|\cdot\|$ to be defined shortly, as required by the foregoing theorem. Also,

$$A^T(\rho - \bar{\rho}) = \Delta(\bar{\rho} - \rho)e. \tag{48}$$

Assume that

$$\left(1 - \frac{1}{40\sqrt{n}\sqrt{\kappa_2(W)}}\right)\mu \leqq \mu' \leqq \mu.$$

Define $f: \mathscr{X} \to R^n$ by

$$f(w) = (X + \hat{X})\Delta(\pi)e - \mu' We. \tag{49}$$

Let $\bar{\xi}' = \bar{\xi} - Df(\bar{\xi})^{-1} f(\bar{\xi})$. We now set out to show that

$$\|\bar{\xi}' - \xi'\|' \leqq \frac{1}{20\| W^{-1/2}\|_2} = \delta, \tag{50}$$

where we must first define the norms $\|\cdot\|$ and $\|\cdot\|'$ appropriately. We do this as follows.

Define

$$\|w\| = \frac{1}{\mu}\|Df(\xi)w\|_{W^{-1}}, \tag{51}$$

where

$$Df(w) = [(X + \hat{X})A^T \,|\, \Delta(\pi)], \qquad Df(\xi)w = Df(\xi)\begin{bmatrix}\pi \\ x\end{bmatrix} = (\Omega + \hat{X})A^T\pi + \Delta(\rho)x.$$

Therefore, for $w = (\pi, x) \in \mathscr{X}$,

$$\|w\| = \frac{1}{\mu}\|(\Omega + \hat{X})A^T\pi + X\Delta(\rho)e\|_{W^{-1}}. \tag{52}$$

Simple manipulations show that our definition of the norm (52) reduces to

$$\|w\| = \frac{1}{\mu}[\|(\Omega + \hat{X})A^T\pi\|^2_{W^{-1}} + \|X\Delta(\rho)e\|^2_{W^{-1}}]^{1/2}. \tag{53}$$

For $\mu' > 0$ define $\xi' = (\rho', \omega')$, Ω', and $\|\cdot\|'$ in an analogous manner.

Now, from the definition of β, see (38) and then, using (51), (49), and (45), we have

$$\beta = \beta(\xi) = \|Df(\xi)^{-1}f(\xi)\| \leqq \frac{(\mu - \mu')}{\mu}\sqrt{n}\,\|W^{1/2}\|_2.$$

Then (44) gives

$$\beta \leqq \frac{1}{40\|W^{-1/2}\|_2}. \tag{54}$$

Next we turn to the second constant γ.

Let $w^{[i]} = (\pi^{[i]}, x^{[i]})$, $i = 1, 2$ denote two arbitrary vectors in $\mathscr{X}$. The operator $D^2f(\xi)$ maps $(w^{[1]}, w^{[2]})$ to the vector $X^{[1]}A^T\pi^{[2]} + X^{[2]}A^T\pi^{[1]}$. Using the norm definition (51) it follows that $Df(\xi)^{-1}D^2f(\xi)$ maps $(w^{[1]}, w^{[2]})$ to a vector of $\|\cdot\|$-length at most

$$\frac{1}{\mu}\|X^{[1]}A^T\pi^{[2]}\|_{W^{-1}} + \frac{1}{\mu}\|X^{[2]}A^T\pi^{[1]}\|_{W^{-1}}.$$

Consider each of the terms in the foregoing expression:

$$\|X^{[1]}A^T\pi^{[2]}\|_{W^{-1}} = \left\| X^{[1]}\left[\frac{1}{\mu}(\Omega + \hat{X})\Delta(\rho)\,W^{-1}\right]A^T\pi^{[2]}\right\|_{W^{-1}} \quad \text{using (46)}$$

$$= \frac{1}{\mu}\|W^{-1/2}X^{[1]}[(\Omega + \hat{X})\Delta(\rho)\,W^{-1}]A^T\pi^{[2]}\|_2$$

$$\leqq \frac{1}{\mu}\|W^{-1/2}\|_2\|W^{-1/2}X^{[1]}\Delta(\rho)\,W^{-1/2}(\Omega + \hat{X})A^T\pi^{[2]}\|_2$$

$$\leqq \mu\|W^{-1/2}\|_2\|w^{[1]}\|\,\|w^{[2]}\| \quad \text{from norm inequalities and (53).}$$

Hence

$$\frac{1}{\mu}\|X^{[1]}A^T\pi^{[2]}\|_{W^{-1}} \leqq \|W^{-1/2}\|_2\|w^{[1]}\|\,\|w^{[2]}\|.$$

Similarly,

$$\frac{1}{\mu}\|X^{[2]}A^T\pi^{[1]}\|_{W^{-1}} \leqq \|W^{-1/2}\|_2\|w^{[1]}\|\,\|w^{[2]}\|.$$

Hence

$$\frac{1}{\mu}\|X^{[1]}A^T\pi^{[2]}\|_{W^{-1}} + \frac{1}{\mu}\|X^{[2]}A^T\pi^{[1]}\|_{W^{-1}} \leqq 2\|W^{-1/2}\|_2\|w^{[1]}\|\,\|w^{[2]}\|.$$

Then using the usual operator norm definition, given at the beginning of this section, we obtain

$$\tfrac{1}{2}\|Df(\xi)^{-1}D^2f(\xi)\| \leqq \|W^{-1/2}\|_2. \tag{55}$$

Observe also that $D^kf \equiv 0$ for all $k > 2$. Hence

$$\gamma = \sup_{k \geqq 2}\left\|\frac{1}{k!}Df(\xi)^{-1}D^kf(\xi)\right\|^{1/(k-1)} \leqq \|W^{-1/2}\|_2. \tag{56}$$

Combining (54) and (56) we have

$$\beta\gamma \leqq \frac{1}{40}. \tag{57}$$

Note also that $\beta \leqq \frac{1}{2}\delta$. Then the theorem quoted at the beginning of this section implies that

$$\|\bar{\xi}' - \xi'\| \leqq \frac{7}{8}\left(\frac{1}{20\|W^{-1/2}\|_2}\right) \quad \text{and} \quad \|\xi - \xi'\| \leqq \frac{3}{80\|W^{-1/2}\|_2}. \tag{58}$$

Finally, we seek a bound on $\|\bar{\xi}' - \xi'\|'$. Let us first apply Proposition 2 to the evaluation of $\|\Delta(\rho')\Delta(\rho)^{-1}\|_{W^{-1}}$.

$$\begin{aligned}
\|\Delta(\rho')\Delta(\rho)^{-1}\|_{W^{-1}} &= \|\Delta(\rho')\Delta(\rho)^{-1}\|_2 \\
&= \|I + \Delta(\rho)^{-1}\Delta(\rho' - \rho)\|_2 \leqq \|I\|_2 + \|\Delta(\rho)^{-1}\Delta(\rho' - \rho)\|_2 \\
&= 1 + \frac{1}{\mu}\|W^{-1}(\Omega + \hat{X})\Delta(\rho' - \rho)\|_2 \quad \text{using (46)} \\
&\leqq 1 + \frac{1}{\mu}\|W^{-1/2}\|_2\|W^{-1/2}(\Omega + \hat{X})\Delta(\rho' - \rho)e\|_2 \quad \text{by Proposition 1} \\
&= 1 + \frac{1}{\mu}\|W^{-1/2}\|_2\|(\Omega + \hat{X})A^T(\rho' - \rho)\|_{W^{-1}}.
\end{aligned}$$

Then using (53) applied to $\|\xi' - \xi\|$, where $(\xi' - \xi) = (\rho' - \rho, \omega' - \omega)$, gives

$$\|\Delta(\rho')\Delta(\rho)^{-1}\|_{W^{-1}} \leqq 1 + \|W^{-1/2}\|_2\|\xi' - \xi\|. \tag{59}$$

Similarly,

$$\|(\Omega' + \hat{X})(\Omega + \hat{X})^{-1}\|_{W^{-1}} \leqq 1 + \|W^{-1/2}\|_2\|\xi' - \xi\|. \tag{60}$$

Therefore

$$\begin{aligned}
\|w\|' &= \frac{1}{\mu'}[\|(\Omega' + \hat{X})A^T\pi\|_{W^{-1}}^2 + \|X\Delta(\rho')e\|_{W^{-1}}^2]^{1/2} \\
&\leqq \frac{1}{\mu'}[\|(\Omega' + \hat{X})(\Omega + \hat{X})^{-1}\|_{W^{-1}}^2\|(\Omega + \hat{X})A^T\pi\|_{W^{-1}}^2 \\
&\qquad + \|\Delta(\rho')\Delta(\rho)^{-1}\|_{W^{-1}}^2\|X\Delta(\rho)e\|_{W^{-1}}^2]^{1/2} \\
&\leqq (1 + \|W^{-1/2}\|_2\|\xi' - \xi\|) \\
&\qquad \cdot \left\{\frac{1}{\mu'}[\|(\Omega + \hat{X})A^T\pi\|_{W^{-1}}^2 + \|X\Delta(\rho)e\|_{W^{-1}}^2]^{1/2}\right\} \quad \text{from (59), (60).}
\end{aligned}$$

Thus

$$\begin{aligned}
\|w\|' &\leqq (1 + \|W^{-1/2}\|_2\|\xi' - \xi\|)\|w\|, \\
\|\bar{\xi}' - \xi'\|' &\leqq (1 + \|W^{-1/2}\|_2\|\xi' - \xi\|)\|\bar{\xi}' - \xi'\|.
\end{aligned} \tag{61}$$

Using (58), we obtain

$$\|\bar{\xi}' - \xi'\|' \leqq \frac{1}{20}\frac{1}{\|W^{-1/2}\|_2} = \delta \quad \text{(see (47)).}$$

This completes the proof of the claim (50).

4.1. Feasibility of $(\bar{\rho}, \bar{\omega}+\hat{x})$.

$$\frac{1}{20\|W^{-1/2}\|_2} \geqq \|\bar{\xi}-\xi\|$$

$$\geqq \frac{1}{\mu}[\|(\Omega+\hat{X})A^T(\bar{\rho}-\rho)\|_{W^{-1}}]$$

$$= \frac{1}{\mu}\|W^{-1/2}[(\Omega+\hat{X})\Delta(\bar{\rho})e-\mu We]\|_2 \quad \text{using (48) and (45).}$$

Thus

$$\frac{1}{20\|W^{-1/2}\|_2} \geqq \frac{1}{\mu}\|W^{-1/2}(\Omega+\hat{X})\Delta(\bar{\rho})e-\mu W^{1/2}e\|_2. \tag{62}$$

Suppose that $\Delta(\bar{\rho})$ contains a *negative* component corresponding, say, to index k. Then (62) implies that $1/(20\|W^{-1/2}\|_2) \geqq W_{kk}^{1/2}$. But $\|W^{-1/2}\|_2 = 1/(\min_i (W_{ii}^{1/2}))$. Thus $(1/20)\min_i (W_{ii}^{1/2}) \geqq W_{kk}^{1/2}$, giving a contradiction. Therefore $\bar{\rho}$ is dual feasible. A similar argument establishes that $\bar{\omega}+\hat{x}$ is primal feasible.

4.2. Duality gap. We show that

$$|b^T\bar{\rho}-c^T(\bar{\omega}+\hat{x})| < 2\mu n\|W\|_2.$$

First consider the duality gap at $(\rho, \omega+\hat{x})$, namely, $|b^T\rho - c^T(\omega+\hat{x})|$. Since $(\omega+\hat{x})$ is feasible for the primal, $A(\omega+\hat{x}) = b$. Thus

$$|b^T\rho-(\omega+\hat{x})^Tc| = |(\omega+\hat{x})^TA^T\rho - e^T(\Omega+\hat{X})c|$$

$$= |e^T(\Omega+\hat{X})A^T\rho - e^T[(\Omega+\hat{X})c-(\Omega+\hat{X})\Delta(\rho)+\mu We]| \quad \text{by (45)}$$

$$= |e^T(\Omega+\hat{X})(A^T\rho-c-\Delta(\rho))-\mu e^TWe| = \mu e^TWe$$

$$\leqq \mu n\|W\|_2.$$

Also,

$$|b^T(\bar{\rho}-\rho)| = |e^T(\Omega+\hat{X})A^T(\bar{\rho}-\rho)| \leqq \|e\|_W\|(\Omega+\hat{X})A^T(\bar{\rho}-\rho)\|_{W^{-1}},$$

using $|a^Tb| \leqq \|a\|_W\|b\|_{W^{-1}}$. Thus $|b^T(\bar{\rho}-\rho)| \leqq \sqrt{e^TWe}(\mu\|\bar{\xi}-\xi\|)$, using the first relation of the preceding subsection on feasibility (§ 4.1). Thus

$$|b^T(\bar{\rho}-\rho)| \leqq \frac{\mu}{20}\frac{\sqrt{e^TWe}}{\|W^{-1/2}\|_2}.$$

Now $\|W^{-1/2}\|_2\|W^{1/2}\|_2 \geqq 1$ implies that $1/(\|W^{-1/2}\|_2) \leqq \|W^{1/2}\|_2$. Therefore

$$|b^T(\bar{\rho}-\rho)| \leqq \frac{\mu}{20}\sqrt{e^TWe}\,\|W^{1/2}\|_2 \leqq \frac{\mu}{20}\sqrt{n}\,\|W\|_2.$$

Similarly,

$$|c^T(\bar{\omega}-\omega)| \leqq \frac{\mu}{20}\sqrt{n}\,\|W\|_2.$$

Hence

$$|b^T\bar{\rho}-c^T(\bar{\omega}+\hat{x})| \leqq 2\mu n\|W\|_2.$$

This result, along with (44), then implies, in the usual way, an $O(\sqrt{n}L)$ iteration bound from all starting points with a fixed prescribed bound on the associated condition number $\kappa_2(W)$. (Here L denotes the number of bits required to specify the problem.)

5. Concluding remarks. In conclusion, we reiterate the main theme of this paper. The field of mathematical programming is at the threshold of a new generation of large-scale mathematical programming systems that must effectively integrate vertex-following (simplex) and interior-point techniques. The homotopy principle and Newton's method provide an extremely rich and powerful unifying framework for deriving algorithms and implementing them effectively within such mathematical programming systems. This requires a careful hybridizing of the strategies discussed in § 3 applied to the canonical system of § 3.5, together with the use of appropriate iterative techniques of computational linear algebra. Such research is currently being undertaken.

In § 4 we also studied a particular elevator predictor, Newton corrector homotopy algorithm and showed that every pair of feasible interior primal and dual points determines a path to the optimal solution characterized by a "condition" $\kappa_2(W)$ and an associated "complexity," which is defined in terms of the quantities n, L, and $\kappa_2(W)$. A balanced choice can be made between path following and condition improvement, suggesting a novel approach to algorithm development. The theoretical (and practical) study of Euler predictor, Newton corrector homotopy algorithms in this setting, as well as extensions given in § 3.5 for starting homotopy algorithms from infeasible points, would also be worthwhile.

Acknowledgments. I thank J. Renegar for helpful interaction on the material of § 4 and its algorithmic implications. I am also grateful to the referees and editor for their helpful comments.

REFERENCES

[1] I. ADLER, N. KARMARKAR, M. G. C. RESENDE, AND G. VEIGA (1986), *An implementation of Karmarkar's algorithm for linear programming*, Report No. ORC 86-8, Operations Research Center, University of California, Berkeley, CA.

[2] D. A. BAYER AND J. C. LAGARIAS (1986), *The nonlinear geometry of linear programming.* I: *Affine and projective scaling trajectories*, II: *Legendre transform coordinates and central trajectories*, AT&T Bell Laboratories, Murray Hill, NJ, preprint; Trans. Amer. Math. Soc., to appear.

[3] G. B. DANTZIG (1963), *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ.

[4] I. I. DIKIN (1967), *Iterative solution of problems of linear and quadratic programming*, Soviet Math. Dokl., 8, pp. 674-675.

[5] B. C. EAVES (1979), *A view of complementary pivot theory*, in Constructive Approaches to Mathematical Models, C. Coffman and G. Fix, eds., Academic Press, New York, pp. 153-170.

[6] A. V. FIACCO AND G. P. MCCORMICK (1968), *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York.

[7] R. FRISCH (1956), *La résolution des problemes de programme linéaire par la méthode du potential logarithmique*, Cahiers du Seminaire D'Econometrie, 4, pp. 7-20; Discussion, pp. 20-23.

[8] C. B. GARCIA AND W. I. ZANGWILL (1981), *Pathways to Solutions, Fixed Points and Equilibria*, Prentice-Hall, Englewood Cliffs, NJ.

[9] P. E. GILL, W. MURRAY, M. A. SAUNDERS, J. A. TOMLIN, AND M. H. WRIGHT (1986), *On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method*, Math. Programming, 36, pp. 183-209.

[10] N. KARMARKAR (1984), *A new polynomial-time algorithm for linear programming*, Combinatorica, 4, pp. 373-395.

[11] M. KOJIMA, S. MIZUNO, AND A. YOSHISE (1987), *A primal-dual interior point method for linear programming*, Res. Report No. B-188, Department of Information Sciences, Tokyo Institute of Technology, Tokyo, Japan.

[12] C. E. LEMKE (1968), *On complementary pivot theory*, in Mathematics of the Decision Sciences, G. B. Dantzig and A. F. Veinott, eds., American Mathematical Society, Providence, RI.

[13] N. MEGIDDO (1986), *Pathways to the optimal set in linear programming*, Res. Report RJ 5295, IBM Almaden Research Center, San Jose, CA. Also in Progress in Mathematical Programming: Interior-Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, New York, 1989, pp. 131–158.

[14] R. D. C. MONTEIRO, I. ADLER, AND M. G. C. RESENDE (1988), *A polynomial-time primal-dual affine scaling algorithm for linear and convex quadratic programming*, Report ESRC 88-8, Department of Industrial Engineering and Operations Research, University of California, Berkeley, CA.

[15] J. L. NAZARETH (1986), *Homotopy techniques in linear programming*, Algorithmica, 1, pp. 529–535.

[16] ——— (1987), *Pricing criteria in linear programming*, Report PAM-382, Center for Pure and Applied Mathematics, University of California, Berkeley, CA. Also in Progress in Mathematical Programming: Interior Point and Related Methods, N. Megiddo, ed., Springer-Verlag, Berlin, New York, 1989, pp. 105–129.

[17] ——— (1989), *The homotopy principle and algorithms for linear programming*, presented at the IIASA Workshop on Nonstandard Optimization Methods and Related Topics, August 1–4, 1989, International Institute for Applied Systems Analysis, Laxenburg, Austria. Also appeared as Tech. Report 90-1, Department of Pure and Applied Mathematics, Washington State University, Pullman, WA.

[18] J. RENEGAR (1988), *A polynomial-time algorithm, based on Newton's method, for linear programming*, Math. Programming, 40, pp. 59–93.

[19] J. RENEGAR AND M. SHUB (1988), *Simplified complexity analysis for Newton* LP *methods*, Report No. 807, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY.

[20] C. ROOS AND D. DEN HERTOG (1989), *A polynomial method of approximate weighted centers for linear programming*, Report 99-13, Faculty of Technical Mathematics and Informatics, Delft University of Technology, Delft, the Netherlands.

[21] S. SMALE (1986), *Algorithms for solving equations*, in Proc. Internat. Congress of Mathematicians, University of California, Berkeley, CA.

[22] G. SONNEVEND (1985), *An analytic center for polyhedrons and new classes of global algorithms for linear (smooth, convex) programming*, in Proc. 12th Internat. Federation of Information Processing Societies Conference on System Modelling and Optimization, Budapest, Hungary.

[23] G. W. STEWART (1988), *On scaled projections and pseudo-inverses*, Report TR-2026, Computer Science Technical Report Series, University of Maryland, College Park, MD.

[24] P. TSENG (1989), *Complexity analysis of a linear complementarity algorithm based on a Lyapunov function*, Report No. P-1884, Laboratory for Informatics and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA.

[25] R. J. VANDERBEI AND J. C. LAGARIAS (1988), *I. I. Dikin's convergence result for the affine-scaling algorithm*, preprint, AT&T Bell Laboratories, Murray Hill, NJ.

[26] L. T. WATSON (1986), *Numerical linear algebra aspects of globally convergent homotopy methods*, Tech. Report 86-7, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 333–357, August 1991

003

A CURVILINEAR SEARCH USING TRIDIAGONAL SECANT UPDATES FOR UNCONSTRAINED OPTIMIZATION

J. E. DENNIS, JR.[†], N. ECHEBEST[‡], M. T. GUARDARUCCI[‡], J. M. MARTÍNEZ[§], H. D. SCOLNIK[¶], AND C. VACCHINO[‡]

Abstract. The idea of doing a curvilinear search along the Levenberg–Marquardt path $s(\mu) = -(H+\mu I)^{-1}g$ always has been appealing, but the cost of solving a linear system for each trial value of the parameter μ has discouraged its implementation. In this paper, an algorithm for searching along a path which includes $s(\mu)$ is studied. The algorithm uses a special inexpensive QT_cQ^T to QT_+Q^T Hessian update which trivializes the linear algebra required to compute $s(\mu)$. This update is based on earlier work of Dennis and Marwil and Martínez on least-change secant updates of matrix factors. The new algorithm is shown to be local and q-superlinearly convergent to stationary points, and to be globally q-superlinearly convergent for quasi-convex functions. Computational tests are given that show the new algorithm to be robust and efficient.

Key words. unconstrained optimization, trust regions, curvilinear search, Levenberg–Marquardt, factor updating, least change secant methods

AMS(MOS) subject classification. 65K

1. Introduction.

In this paper, we consider iterative methods for solving the smooth unconstrained minimization problem:

$$\min_x f(x); \qquad f : \Omega \subset \mathbb{R}^n \to \mathbb{R}; \qquad f \in C^1(\Omega)\,,$$

for Ω open in $\mathbb{R}^n$. We denote $g(x) = \nabla f(x)$ for all $x \in \Omega$. We will use the ℓ_2 norm whenever another norm is not indicated.

Our methods are based on the common notion of choosing a trial step from the current iterate x_c to the next iterate x_+ based on a local quadratic model of $f(x_c+s) - f(x_c)$ of the form:

$$q_c(s) \equiv g_c^T s + \frac{1}{2} s^T H_c s, \quad \text{where } g_c = \nabla f(x_c) \quad \text{and} \quad H_c = H_c^T. \tag{1}$$

Our methods belong to a class often called curvilinear search methods, and the curvilinear path we search along is the same one in $\mathbb{R}^n$ from which the trust-region method based on the same model would choose its step. The major difference from trust-region methods is that, even if we eventually choose the same trial step, we do our search based on the "Levenberg–Marquardt" parameter rather than on the length of the step. Methods based on other curvilinear paths have been published, but since none are in general use, we omit any comparative discussion. Most relevant is that Schramm and

* Received by the editors January 11, 1990; accepted for publication (in revised form) January 4, 1991.

† Mathematical Sciences Department, Rice University, Houston, Texas 77251-1892. This work was begun under a Fulbright Fellowship to Argentina. This research was partially supported by Air Force Office of Scientific Research grants AFOSR-89-0363, DOE/ER/25017-3, DAAL03-90-0093, and National Science Foundation grant DMS-8903751.

‡ Departamento de Matemática, Universidad de La Plata, La Plata, Buenos Aires, Argentina.

§ Universidade Estadual de Campinos, Campinos, Brasil. This work was done while the author visited the Mathematical Sciences Department, Rice University, and was supported by a fellowship from FAPESP, Brasil.

¶ Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Argentina.

Zowe [11] in their B-T algorithm for nonsmooth optimization search the analogous curve.

The key to the practicality of the particular method we test is that we build the local model (1) in a form that trivializes the linear algebra needed to compute any trial step along the search path. For example, standard approaches would require a Cholesky factorization at each trial step, but we need only solve a tridiagonal system and do two matrix-vector products.

This paper is organized as follows: §2 contains a global convergence analysis in which we assume that the sequence of model Hessians is bounded, but we do not specify how the Hessians are to be chosen. We define the set from which a trial step must be chosen that satisfies an Armijo criterion. We show that there are steps in the set that satisfy the sufficient decrease criterion, but we do not specify how the step is to be found.

In §3, we assume that $\nabla^2 f$ is Lipschitz continuous on Ω, and we present a new least-change secant method for defining H_+ from H_c and apply the results of §2 to the resulting algorithm. This method is in the spirit of [2], [7], and [5] in that there is never any need to form H_+. Instead, H_c is held in the form $Q_0 T_c Q_0^T$, Q_0 orthogonal, T_c tridiagonal; and $H_+ = Q_0 T_+ Q_0^T$ is defined by doing a sparse symmetric secant update of T_c to get T_+.

In §4, we validate the new update by giving a local convergence analysis of the corresponding full step quasi-Newton method to stationary points of f. In §5, we add a convexity assumption on f and prove that the particular method from §3 that always tries the Newton step first when H_c is positive definite is globally q-superlinearly convergent. This order of convergence result is no better than we could prove if we did not do the updates, but the updates cost a low multiple of n, and they are certainly worthwhile computationally, as is shown in §7.3. Section 6 discusses an implementation and §7 gives some numerical results for a particular method from §3.

2. The general algorithm: Global convergence. In this section we state a general algorithm of the type studied here. We make the algorithm only as specific as necessary to prove a global convergence result.

Given $x \in \Omega$, H a symmetric $n \times n$ matrix, $\lambda_1 = \lambda_1(H)$ the smallest eigenvalue of H, V_1 the corresponding eigenspace, we define a curve parameterized by mu:

$$\Gamma_1(x, H) = \{x - (H + \mu I)^{-1} g(x) : 0 \leq \mu > -\lambda_1\} .$$

If $g(x) \notin V_1^{\perp}$, or if $\lambda_1 > 0$, we define $\Gamma(x, H) = \Gamma_1(x, H)$. Otherwise, we choose $v \in V_1$, $v \neq 0$, and we define a curve parameterized by mu:

$$\Gamma(x, H) = \Gamma_1(x, H) \cup \Gamma_2(x, H) ,$$

where

$$\Gamma_2(x, H) = \{x - (H - \lambda_1 I)^{+} g(x) + \mu v : \mu \in \mathbb{R}\} .$$

The following lemma, which follows from Gay [4] and Moré and Sorensen [8], gives a geometrical meaning to $\Gamma(x, H)$. It shows that if $\lambda_1 \leq 0$ and if $g(x) \in V_1^{\perp}$, then any $v \in V_1$ gives the same result for the quadratic. In our implementation, we always choose trial steps that stand in the same relation to the current iterate that z has to x in the hypotheses of the lemma. However, we have no need to be so specific in order to prove global convergence in the next section.

LEMMA 2.1. *Let $x \in \Omega$, $z \in \Gamma(x, H)$. Then z is a minimizer of*

$$q(w) = \tfrac{1}{2}(w-x)^T H(w-x) + g(x)^T(w-x) \quad \textit{subject to } \|w-x\| \le \|z-x\|\,,$$

and the direction from x to z is a descent direction for q. Furthermore, assume $z \in \Gamma_1(x, H)$; then z is the unique minimizer. If $0 < \delta \le \|z-x\|$, then there is a unique $w \in \Gamma(x, H)$ such that $\|w-x\| = \delta$. Also, $w \in \Gamma_1(x, H)$.

Proof. This is just a slight restatement of a standard result of Gay[4] and Sorensen. For example, see Lemma 2.3 of Moré and Sorensen [8]. □

The following algorithm describes the way of obtaining a new approximation x_+ to the minimizer of f, starting from a current approximation $x_c \in \Omega$ such that $g_c \neq 0$ and using a current Hessian approximation H_c. A large positive number Δ is used to bound the steplength, and Δ_c and $\bar{\Delta}_c$ are constants needed in the convergence proof. The algorithm parameters $\alpha \in (0, \frac{1}{2}), \beta \in (0,1)$ are used to guarantee sufficient decrease. We use $\alpha = 10^{-4}$ and $\beta =$ macheps.

ALGORITHM 2.1.

Given H_c, x_c;
If $\lambda_1(H_c) \le 0$; Then $\Delta_c = \bar{\Delta}_c = \Delta$;
  Else $s_c^N = -H_c^{-1} g(x_c)$; $\Delta_c = \bar{\Delta}_c = \min\{\Delta, 1/\beta \|s_c^N\|\}$;
Set $\bar{x} = x_c$;
While ($\bar{x} = x_c$ or $f(\bar{x}) > f(x_c) + \alpha g(x_c)^T(\bar{x} - x_c)$) DO
    Choose $\bar{x} \in \Gamma(x_c, H_c)$ such that $\beta^2 \Delta_c \le \|\bar{x} - x_c\| \le \Delta_c$;
    $\Delta_c = \Delta_c/2$;
  ENDO;
  Set $x_+ = \bar{x}$;

Remark. Obviously, the efficiency of Algorithm 2.1 depends on the way $\bar{x}$ is selected. "Choose" is a very ambiguous word that we use deliberately to show that many strategies are possible.

Let us now prove that, given x_c, H_c, with $g_c = g(x_c) \neq 0$, Algorithm 2.1 is always able to finish by finding a point $\bar{x}$ which satisfies the sufficient decrease condition

$$f(\bar{x}) \le f_c + \alpha g_c^T(\bar{x} - x_c). \tag{2}$$

THEOREM 2.2. *After a finite number of DO loop executions, Algorithm* 2.1 *obtains a point $\bar{x} = x_+$ that satisfies* (2).

Proof. We only need to prove that, if $\|\bar{x} - x_c\|$ is small enough and $\bar{x} \in \Gamma(x_c, H_c)$, then (2) is satisfied. Using Lemma 2.1, it is easy to see that

$$\lim_{\substack{\bar{x} \to x_c \\ \bar{x} \in \Gamma(x_c, H_c)}} \frac{\bar{x} - x_c}{\|\bar{x} - x_c\|} = \lim_{\substack{\bar{x} \to x_c \\ \bar{x} \in \Gamma_1(x_c, H_c)}} \frac{\bar{x} - x_c}{\|\bar{x} - x_c\|} = \frac{-g(x_c)}{\|g(x_c)\|}\,, \tag{3}$$

since if $\|\bar{x} - x_c\|$ is small enough, then $\bar{x} \in \Gamma_1(x_c, H_c)$. Therefore, using (3) and the Mean Value Theorem, we have

$$\frac{f(\bar{x}) - f(x_c)}{\|\bar{x} - x_c\|} = \frac{g(x_c + \xi(\bar{x} - x_c))^T(\bar{x} - x_c)}{\|\bar{x} - x_c\|} \quad \text{with } \xi \in (0,1)\,.$$

Hence,

$$\lim_{\substack{\bar{x}\to x_c \\ \bar{x}\in\Gamma(x_c,H_c)}} \frac{f(\bar{x})-f(x_c)}{\|\bar{x}-x_c\|} = g(x_c)^T \lim_{\substack{\bar{x}\to x_c \\ \bar{x}\in\Gamma_1(x_c,H_c)}} \frac{\bar{x}-x_c}{\|\bar{x}-x_c\|} = -\|g(x_c)\|$$

$$\leq -\alpha\|g(x_c)\| \leq +\alpha\frac{g_c^T(\bar{x}-x_c)}{\|\bar{x}-x_c\|}$$

for any $\bar{x} \neq x_c$, and the required result follows from this inequality. □

We now give a result that we need to prove global convergence of Algorithm 2.1.

LEMMA 2.3. *Assume that* $\|H_k\| \leq B$ *for* $k = 0, 1, 2, \cdots$ *and* $\lim_{k\to\infty} x_k = x_*$ *with* $g(x_*) \neq 0$. *Let* $\{\bar{x}_k\}$ *be any sequence such that* $\bar{x}_k \in \Gamma(x_k, H_k)$, $\lim_{k\to\infty} \|\bar{x}_k - x_k\| = 0$. *Then there exists a subsequence* $\{\bar{x}_{k_j} - x_{k_j}\}$ *such that, for this subsequence,*

$$\lim_{j\to\infty} \frac{\bar{x}_{k_j}-x_{k_j}}{\|\bar{x}_{k_j}-x_{k_j}\|} = \frac{-g(x_*)}{\|g(x_*)\|} .$$

Proof. Let $\{H_k\}_{k\in K_1}$ be a convergent subsequence of $\{H_k\}$. Then for some H,

$$\lim_{k\in K_1} H_k = H , \qquad \|H\| \leq B .$$

For $k \in K_1$ let us write

$$H_k = Q_k D_k Q_k^T , \tag{4}$$

where $D_k = \operatorname{diag}(\lambda_1(H_k), \cdots, \lambda_n(H_k))$, $\lambda_1(H_k) \leq \cdots \leq \lambda_n(H_k)$. By the continuity property of eigenvalues (see Wilkinson [12, p. 63] or Ostrowski [9, p. 225]), we have:

$$\lim_{k\in K_1} \lambda_i(H_k) = \lambda_i(H) , \qquad i = 1, \cdots, n,$$

where $\lambda_i(H)$, $i = 1, \cdots, n$ are the eigenvalues of H in increasing order. Now, the matrices $\{Q_k\}_{k\in K_1}$ are contained in a compact set of $\mathbb{R}^{n\times n}$. Therefore, there exists a convergent subsequence $\{Q_k\}_{k\in K_2}$, $K_2 \subset K_1$ such that

$$\lim_{k\in K_2} Q_k = Q ,$$

and Q is an orthogonal $n \times n$ matrix. Hence, taking limits in (4) for $k \in K_2$, we have:

$$H = QDQ^T ,$$

where $D = \operatorname{diag}(\lambda_1(H), \cdots, \lambda_n(H))$, $Q = (v_1, \cdots, v_n)$. Now, $g(x_*) \neq 0$, so there exists $m \in \{1, \cdots, n\}$ such that

$$g(x_*)^T v_m \neq 0 . \tag{5}$$

Therefore, there exists $\mu > -\lambda_1$ such that

$$\frac{|g(x_*)^T v_m|}{\lambda_m + \mu} \geq \frac{\gamma}{2} \left(\equiv \frac{1}{2} \min\left\{ \frac{|g(x_*)^T v_m|}{\lambda_m - \lambda_1}, 1 \right\} \right) . \tag{6}$$

Hence, taking limits for $k \in K_2$, we have, for large enough $k \in K_2$,

$$\frac{|g(x_k)^T v_m^k|}{\lambda_m(H_k) + \mu} \geq \frac{\gamma}{4} . \tag{7}$$

But,

$$x_k - (H_k + \mu I)^{-1} g(x_k) \in \Gamma_1(x_k, H_k),$$

and

$$\| - (H_k + \mu I)^{-1} g(x_k)\| \geq \frac{|g(x_k)^T v_m^k|}{\lambda_m(H_k) + \mu}.$$

Therefore, for large enough $k \in K_2$, by Lemma 2.1 and (7) there exists $z_k \in \Gamma_1(x_k, H_k)$ such that

$$\|x_k - z_k\| = \frac{\gamma}{4}. \tag{8}$$

Hence, since $\lim_{k\to\infty} \|\bar{x}_k - x_k\| = 0$, Lemma 2.1 and (8) imply that $\bar{x}_k \in \Gamma_1(x_k, H_k)$ for large enough $k \in K_2$ (say, $k \in K_3$).

We now want to prove that $\lim_{k\to\infty} \mu_k = \infty$. We proceed by contradiction. Assume that $\mu_k \leq \mu_0 < \infty$ for $k \in K_4 \subset K_3$. Then, $\bar{x}_k \in \Gamma_1(x_k, H_k)$ for $k \in K_4$, so for $Q_k = (v_1^k, \cdots, v_n^k)$,

$$\begin{aligned}
\|\bar{x}_k - x_k\|^2 &= \| - (H_k + \mu_k I)^{-1} g(x_k)\|^2 = \| - Q_k (D_k + \mu_k I)^{-1} Q_k^T g(x_k)\|^2 \\
&= \|(D_k + \mu_k I)^{-1} Q_k^T g(x_k)\|^2 \\
&= \left(\frac{g(x_k)^T v_1^k}{\lambda_1(H_k) + \mu_k}\right)^2 + \cdots + \left(\frac{g(x_k)^T v_n^k}{\lambda_n(H_k) + \mu_k}\right)^2 \qquad (9) \\
&\geq \left(\frac{g(x_k)^T v_1^k}{\lambda_1(H_k) + \mu_0}\right)^2 + \cdots + \left(\frac{g(x_k)^T v_n^k}{\lambda_n(H_k) + \mu_0}\right)^2.
\end{aligned}$$

But the limit of the right-hand side of (9) when $k \to \infty$ is clearly a nonzero positive number, therefore $\|\bar{x}_k - x_k\|^2$ is bounded away from zero if $k \in K_4$ is large enough, contradicting the hypothesis. Hence, $\lim_{k\in K_3} \mu_k = \infty$. Therefore, we may write

$$\begin{aligned}
\frac{\bar{x}_k - x_k}{\|\bar{x}_k - x_k\|} &= \frac{-(H_k + \mu_k I)^{-1} g(x_k)}{\| - (H_k + \mu_k I)^{-1} g(x_k)\|} \\
&= \frac{-(H_k/\mu_k + I)^{-1} g(x_k)}{\|(H_k/\mu_k + I)^{-1} g(x_k)\|},
\end{aligned}$$

and the thesis follows for the subsequence indexed by K_3 using boundedness of $\{H_k\}$ and $\lim_{k\in K_3} \mu_k = \infty$. □

Now we are able to prove the following global convergence theorem. Note that we do not assume that $\nabla^2 f(x_k)$ exists, much less that H_k approximates it well.

THEOREM 2.4. *Assume that* $\|H_k\| \leq B$ *for* $k = 0, 1, 2, \cdots$, $x_0 \in \Omega$ *and* x_{k+1}, $k = 0, 1, 2, \cdots$ *is obtained from Algorithm* 2.1. *Let* $x_* \in \Omega$ *be a limit point of* $\{x_k\}$. *Then* $g(x_*) = 0$.

Proof. Assume that $x_* \in \Omega$, $x_* = \lim_{k\in K_1} x_k$, and $g(x_*) \neq 0$. We consider two possibilities:

(a) Some subsequence of $\{\|x_{k+1} - x_k\|\}_{k\in K_1}$ is bounded away from 0.

(b) $\lim_{k\in K_1} \|x_{k+1} - x_k\| = 0$.

Using Lemma 3.2 of Powell and Yuan [10], we see that

$$g(x_k)^T (x_{k+1} - x_k) \leq -\frac{\|g(x_k)\|^2 \|x_{k+1} - x_k\|}{2\|H_k\| \, \|x_{k+1} - x_k\| + \|g(x_k)\|} \leq -\frac{\|g(x_k)\|^2 \|x_{k+1} - x_k\|}{2B\Delta + \|g(x^k)\|}.$$

Hence, if (a) holds, using (2) and the continuity of ∇f at x_*, we see that $\lim_{k\to\infty} f(x_k) = -\infty$. This contradicts the assumption $x_* \in \Omega$.

Therefore, it remains to analyze (b). Since, in Algorithm 2.1, x_{k+1} is set to $\bar{x}$, which is chosen such that $\|\bar{x} - x_k\| \geq \beta^2\Delta_k/2$, it follows that $\lim_{k\in K_1} \Delta_k = 0$. We consider two possibilities:

(i) For some $K_2 \subset K_1$, $\lim_{k\in K_2} \bar{\Delta}_k = 0$.

(ii) For every $K_3 \subset K_1$, $\lim_{k\in K_3} \bar{\Delta}_k \neq 0$.

If (i) holds, then we can assume for $k \in K_2$ that $\lambda_1(H_k) > 0$ since otherwise $\bar{\Delta}_k = \Delta$. Thus $\bar{\Delta}_k$ is set in Algorithm 2.1 to be the minimum of Δ and $1/\beta\|H_k^{-1}g(x_k)\|$, and it follows that $\lim_{k\in K_2} \| - H_k^+ g(x_k)\| = 0$. But

$$\|g(x_k)\| \leq \|H_k\|\,\|H_k^{-1}g(x_k)\| \leq B\|H_k^{-1}g(x_k)\| .$$

Hence $\lim_{k\in K_2} g(x_k) = 0$ and so, $g(x_*) = 0$, contradicting the initial assumption.

Now consider (ii). It means that the sequence $\{\bar{\Delta}_k\}_{k\in K_1}$ is bounded away from zero. Therefore the first trial point of the algorithm failed to satisfy (2). This is so because $\Delta_k = \bar{\Delta}_k$, the first pass through the DO loop at each iteration, and our working hypothesis at this point is $\lim_{k\in K_1} \Delta_k = 0$. Thus, for all iterations indexed by K_1, there is at least one failed trial point. Let us set the sequence of last failed trials to $\{\bar{x}_k\}_{k\in K_1}$. We have that each $\bar{x}_k$ satisfies

$$\beta^2 2\Delta_k \leq \|\bar{x}_k - x_k\| \leq 2\Delta_k .$$

It follows that

$$\lim_{k\in K_1} \|\bar{x}_k - x_k\| = 0$$

and

$$f(\bar{x}_k) - f(x_k) > -\alpha|g(x_k)^T(\bar{x}_k - x_k)| > -\alpha\|g(x_k)\|\,\|\bar{x}_k - x_k\| .$$

Hence, using the Mean Value Theorem,

$$g(x_k - \xi_k(\bar{x}_k - x_k))^T \frac{(\bar{x}_k - x_k)}{\|\bar{x}_k - x_k\|} > -\alpha\|g(x_k)\| . \tag{10}$$

Now we are under the hypotheses of Lemma 2.3. So, taking limits on both sides of (10) for a suitable subsequence, we obtain

$$g(x_*)^T \left(\frac{-g(x_*)}{\|g(x_*)\|}\right) \geq -\alpha\|g(x_*)\| .$$

But this inequality implies that $\alpha \geq 1$, contradicting the initial hypothesis. Therefore the theorem is proved. □

3. Updating H_k. In §2, we used a uniform bound on $\{\|H_k\|\}$ to obtain a global convergence result for Algorithm 2.1. Algorithm 3.1 proposes a way of updating H_k that under reasonable conditions preserves uniform boundedness of $\{\|H_k\|\}$ and, in addition, incorporates second-order information using secant approximations.

ALGORITHM 3.1. Let $\mathcal{H} \subset \mathbb{R}^{n\times n}$ be a family of symmetric matrices uniformly bounded in norm by M. Let q be a positive integer, $\theta \in (0, \frac{1}{2})$ be a small number, and $\Upsilon \subset \mathbb{R}^{n\times n}$ be the set of tridiagonal symmetric matrices. We now particularize Algorithm 2.1 by specifying that if $k + 1 \equiv 0 \pmod{q}$, then we choose $H_{k+1} \in \mathcal{H}$. Otherwise, we assume that $H_k = Q_k T_k Q_k^T$, $T_k \in \Upsilon$, Q_k orthogonal, and we obtain H_{k+1} by the following steps:

Step 1. Let $s = s_k = Q_k^T(x_{k+1} - x_k)$. If s does not satisfy

$$\sqrt{s_i^2 + s_{i+1}^2} \geq \theta\|s\| \leq \sqrt{s_n^2 + s_1^2}\,, \tag{11}$$

$i = 1, \cdots, n-1$, replace s by any vector satisfying (11) with $\|s\| = \|x_{k+1} - x_k\|$ and $x_k + Q_k s \in \Omega$. We used $s = ((\|x_{k+1} - x_k\|)/\sqrt{n})e$ in our implementation.

Step 2. Define $y = y_k = Q_k^T[g(x_k + Q_k s) - g(x_k)]$. (Observe that $x_k + Q_k s = x_{k+1}$ if s was not replaced at Step 1.)

Step 3. Obtain T_{k+1} as the solution of the problem

$$\min_{\substack{Ts=y \\ T \in \Upsilon}} \|T - T_k\|_F^2.$$

Step 4. $H_{k+1} = Q_{k+1} T_{k+1} Q_{k+1}^T$ with $Q_{k+1} = Q_k$. (Of course, neither H_k nor H_{k+1} need to be formed.)

The solution of (3) may be obtained using the least-change theory for updates by an algorithm which will be described in §5. See, for example, Dennis and Schnabel [3, Chap. 7]. The rest of this section is essentially to prove that the sequence of matrices obtained using Algorithms 2.1 and 3.1 is bounded, and so, that the global convergence Theorem 2.2 holds. Some auxiliary lemmas will be necessary.

LEMMA 3.1. *Let s be such that $s_i^2 + s_{i+1}^2 > 0$; $i = 1, \cdots, n-1$. Define $\tilde{A} \in \mathbb{R}^{n \times (2n-1)}$ as:*

(12)

$$\tilde{A} = \begin{bmatrix} s_1 & s_2/\sqrt{2} & & & & & & & \\ & s_1/\sqrt{2} & s_2 & s_3/\sqrt{2} & & & & & \\ & & & s_2/\sqrt{2} & s_3 & s_4/\sqrt{2} & & & \\ & & & & & \ddots & & & \\ & & & & & & s_{n-2}/\sqrt{2} & s_{n-1} & s_n/\sqrt{2} \\ & & & & & & & s_{n-1}/\sqrt{2} & s_n \end{bmatrix}.$$

Then rank $\tilde{A} = n$.

Proof. Form $\tilde{A}\tilde{A}^T$ and note that it is symmetric and strictly diagonally dominant. □

COROLLARY 3.1. *Under condition* (11), *if $s \neq 0$, rank $\tilde{A} = n$.*

Proof. The proof is trivial using Lemma 3.1. □

Under condition (11) and $s \neq 0$, either $|s_1| \geq (\theta/\sqrt{2})\|s\|$ or $|s_n| \geq (\theta/\sqrt{2})\|s\|$. Let us suppose, without loss of generality, that $|s_n| \geq (\theta/\sqrt{2})\|s\|$ (otherwise the following lemma may be reformulated in an obvious way).

LEMMA 3.2. *Let β_i be the angle between the row $i+1$ of $\tilde{A}$ and the subspace S_i spanned by the i first rows. Assume $s \neq 0$ and* (11). *Then $|\sin \beta_i| \geq (\theta/\sqrt{2})$, $i = 1, \cdots, n-1$.*

Proof. Consider S_i', the subspace of $\mathbb{R}^{(2n-1)}$ formed by the vectors of the form:

$$(z_1, z_2, \cdots, z_{2i}, 0, \cdots, 0)^T\,.$$

Obviously, $S_i \subset S_i'$, $i = 1, \cdots, n-1$.

Let β_i' be the angle between the row $i+1$ of $\tilde{A}$ and s_i'. Then, $|\sin\beta_i| \geq |\sin\beta_i'|$. Now if $i \leq n-2$, then

$$|\sin\beta_i'| = \frac{\sqrt{s_i^2 + s_{i+1}^2/2}}{\sqrt{(s_{i-1}^2/2) + s_i^2 + (s_i^2/2)}}$$

$$\geq \frac{(1/\sqrt{2})\sqrt{s_i^2 + s_{i+1}^2}}{\sqrt{s_{i-1}^2 + s_i^2 + s_{i+1}^2}} \geq \frac{(1/\sqrt{2})\sqrt{s_i^2 + s_{i+1}^2}}{\|s\|} \geq \frac{\theta}{\sqrt{2}}.$$

If $i = n-1$, then

$$|\sin\beta_{n-1}'| = \frac{|s_n|}{\sqrt{(s_{n-1}^2/2) + s_n^2}} \geq \frac{\theta\|s\|/\sqrt{2}}{\|s\|} = \frac{\theta}{\sqrt{2}},$$

so, $|\sin\beta_i| \geq |\sin\beta_i'| \geq (\theta/\sqrt{2})$, $i = 1, \cdots, n-1$. □

LEMMA 3.3. *The product* $\prod = \prod_{i=1}^{n-1} |\sin\beta_i|$ *is invariant under permutations of the rows of* $\tilde{A}$.

Proof. Set $\bar{A} = \binom{\tilde{A}}{H}$ such that $\bar{A}$ is nonsingular. Suppose further that the rows of H are orthogonal and span the orthogonal complement to the rows of $\tilde{A}$. Thus (see [6])

$$\Pi = \frac{|\det\bar{A}|}{W}, \tag{13}$$

where W is the product of the norms of the rows of $\bar{A}$. But the right-hand side of (13) is invariant under permutations of the rows of $\bar{A}$ (and hence, of $\tilde{A}$), so the same happens with Π. □

LEMMA 3.4. *Let* γ_i *be the angle between the row* i *of* $\tilde{A}$ *and the subspace spanned by the other rows of* $\tilde{A}$. *Then* $|\sin\gamma_i| \geq \prod \geq \theta^{n-1}2^{(1-n)/2}$.

Proof. Fix the row i and permute the rows of $\tilde{A}$ so that row i becomes the last one. So $|\sin\gamma_i| = |\sin\beta_{n-1}| \geq \prod = \prod|\sin\beta_i| \geq (\theta/\sqrt{2})^{n-1}$. □

LEMMA 3.5. *Let* $s \neq 0$ *and* $\tilde{A}^+ = \tilde{A}^T(\tilde{A}\tilde{A}^T)^{-1}$. *Then* $\tilde{A}^+ \in \mathbb{R}^{(2n-1)\times n}$. *Let*

$$\tilde{A}^+ = (h_1, \cdots, h_n), \qquad \tilde{A} = \begin{pmatrix} r_1^T \\ \vdots \\ r_n^T \end{pmatrix}. \tag{14}$$

Then $\|h_i\| \leq \dfrac{2^{(n-1)/2}}{\theta^n\|s\|}$.

Proof. Each column h_i of $\tilde{A}^+$ is a linear combination of $r_1, \cdots, r_n$. Moreover $h_i^T r_i = 1$ and $h_i^T r_j = 0$ if $j \neq i$. Let S be the subspace spanned by $\{r_1, \cdots, r_n\}$ (and hence, by $\{h_1, \cdots, h_n\}$). Each r_i may be expressed as

$$r_i = v_i + w_i,$$

where v_i is the projection of r_i on the subspace spanned by $\{r_j, j \neq i\}$ and w_i is the projection of r_i into the line spanned by h_i. So

$$w_i = \frac{h_i^T}{\|h_i\|} r_i \frac{h_i}{\|h_i\|} = \frac{h_i}{\|h_i\|^2}. \tag{15}$$

But

$$\sin\gamma_i = \frac{\|w_i\|}{\|r_i\|}, \quad \text{so} \quad \frac{\|w_i\|}{\|r_i\|} \geq \left(\frac{\theta}{\sqrt{2}}\right)^{n-1}. \tag{16}$$

Thus, by (15) and (16), $1/\|h_i\|\,\|r_i\| \geq \left(\theta/\sqrt{2}\right)^{n-1}$ and hence,

$$\|h_i\| \leq \frac{\left(\theta/\sqrt{2}\right)^{1-n}}{\|r_i\|}.$$

But $\|r_i\| \geq \theta\|s\|$, so

$$\|h_i\| \leq \frac{2^{(n-1)/2}}{\theta^n\|s\|}. \tag{17}$$

□

LEMMA 3.6. *If $s \neq 0$, then for any norm $|\cdot|$ fixed in $\mathbb{R}^{(2n-1)\times n}$, there exists a constant $K_1 = K_1(|\cdot|, \theta, n)$ such that $|\tilde{A}^+| \leq K_1/\|s\|$.*

Proof. The proof is a consequence of (17).

The results above are going to be used in a "vector formulation of the least-change update." Let us write

$$T_k = \begin{bmatrix} a_1^k & b_1^k & & \\ b_1^k & a_2^k & b_2^k & \\ & \ddots & \ddots & \\ & & b_{n-1}^k & a_n^k \end{bmatrix}, \tag{18}$$

$$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & b_2 & \\ & \ddots & \ddots & \\ & & b_{n-1} & a_n \end{bmatrix}. \tag{19}$$

The least-change update is the solution of

$$\min_{\substack{Ts=y \\ T\in\Upsilon}} \|T - T_k\|_F^2. \tag{20}$$

By (18) and (19), (20) may be formulated as follows:

(21)

$$\min \quad (a_1-a_1^k)^2 + 2(b_1-b_1^k)^2 + (a_2-a_2^k)^2 + \cdots + 2(b_{n-1}-b_{n-1}^k)^2 + (a_n-a_n^k)^2$$

$$\text{s.t.} \quad \begin{cases} a_1s_1 + b_1s_2 = y_1 \\ b_1s_1 + a_2s_2 + b_2s_3 = y_2 \\ \qquad\qquad \ddots \\ b_{n-1}s_{n-1} + b_ns_n = y_n \end{cases}.$$

Let us now consider the isomorphism between Υ and $\mathbb{R}^{2n-1}$, which maps

$$T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & b_2 & \\ & \ddots & \ddots & \\ & & b_{n-1} & a_n \end{bmatrix} \overset{\Phi}{\longleftrightarrow} \begin{bmatrix} a_1 \\ b_1 \\ a_2 \\ \vdots \\ b_{n-1} \\ a_n \end{bmatrix} = t. \tag{22}$$

We write $\Phi(T) = t$, $\Phi(T_k) = t_k$, and so on. Therefore, the problem (21) may be written in $\mathbb{R}^{2n-1}$ as:

$$\min_t \|t - t_k\|_G^2 \equiv (t - t_k)^T G(t - t_k) \quad \text{with } G = \begin{pmatrix} 1 & & & & & & \\ & 2 & & & & & \\ & & 1 & & & & \\ & & & 2 & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 2 \\ & & & & & & & 1 \end{pmatrix}$$

such that $A_k t = y$,

$$\text{where} \quad A_k = \begin{pmatrix} s_1 & s_2 & & & \\ & s_1 & s_2 & s_3 & \\ & & \ddots & \ddots & \\ & & & s_{n-1} & s_n \end{pmatrix} \quad \text{and} \quad s_i = (s_k)_i \,. \tag{23}$$

By Lemma 3.1, the matrix $A_k = \tilde{A}_k G^{1/2}$ has full rank, so by straightforward calculations, the solution of (23) is

$$t_{k+1} = t_k - G^{-1/2} \tilde{A}_k^+ (A_k t_k - y), \tag{24}$$

where $\tilde{A}_k$ is defined in (12) and

$$\tilde{A}_k^+ = \tilde{A}_k^T (\tilde{A}_k \tilde{A}_k^T)^{-1} = G^{-1/2} A_k^T (A_k G^{-1} A_k^T)^{-1}. \tag{25}$$

So

$$t_{k+1} = t_k - G^{-1} A_k^T (A_k G^{-1} A_k^T)^{-1} (A_k t_k - y) \,. \tag{26}$$

Therefore

$$\|t_{k+1}\|_G \le \|(I - G^{-1} A_k^T (A_k G^{-1} A_k^T)^{-1} A_k) t_k\|_G + \|G^{-1/2} \tilde{A}_k^+ y\|_G \,.$$

But $(I - G^{-1} A_k^T (A_k G^{-1} A_k^T)^{-1} A_k) t_k$ is the solution of

$$\min \|t - t_k\|_G \quad \text{s.t. } A_k t = 0,$$

so $\|(I - G^{-1} A_k^T (A_k G^{-1} A_k^T)^{-1} A_k) t_k\|_G \le \|t_k\|_G$. Therefore

$$\|t_{k+1}\|_G \le \|t_k\|_G + \|G^{-1/2} \tilde{A}_k^+ y\|_G \,. \tag{27}$$

Now,

$$\|G^{-1/2} \tilde{A}_k^+ y\|_G \le \|G^{-1/2}\|_G \|\tilde{A}_k^+ y\|_G$$

and

$$\begin{aligned} \|\tilde{A}_k^+ y\|_G &= \|y_1 h_1 + \cdots + y_n h_n\|_G \le |y_1| \, \|h_1\|_G + \cdots + |y_n| \, \|h_n\|_G \\ &\le \|y\| (\|h_1\|_G + \cdots + \|h_n\|_G) \,. \end{aligned}$$

But $\|h_1\|_G + \cdots + \|h_n\|_G$ defines a norm in $\mathbb{R}^{(2n-1)\times n}$, so by Lemma 3.6,

$$\|\tilde{A}_k^+ y\|_G \le \frac{K_1 \|y\|}{\|s\|}$$

and so

$$\|G^{-1/2}\tilde{A}_k^+ y\|_G \leq K_2 \frac{\|y\|}{\|s\|} . \tag{28}$$

Now we are able to prove the main result of this section. □

Let

$$L_0 = \{x : f(x) \leq f(x_0)\}.$$

THEOREM 3.7. *Assume that L_0 is bounded and contained in Ω, $f \in C^2(\Omega)$, Ω convex, and that for some $L \geq 0$,*

$$\|\nabla^2 f(x) - \nabla^2 f(w)\| \leq L\|x - w\| \tag{29}$$

for all $x, w \in \Omega$.

Assume that the sequences $\{x_k\}$ and $\{H_k\}$ are generated using Algorithms 2.1 *and* 3.1. *Then the sequence $\{H_k\}$ is bounded by some constant B.*

Proof. Since $\{x_k\}$ is generated by Algorithms 2.1 and 3.1, $\|s\| = \|x_{k+1} - x_k\|$, and Lemma 2.1 implies that $\|s\| = 0$ only if $\{x_k\}$ converges to a stationary point in finitely many steps. Using (29), we have

$$\|y - \nabla^2 f(x_k)s\| \leq \frac{L}{2}\|s\|^2 .$$

Since $\|\nabla^2 f(x)\|$ is bounded uniformly on L_0 by continuity, and since $\{x_k\}$ contained in L_0 implies that $\|s\|$ is uniformly bounded,

$$\|y\| \leq \|\nabla^2 f(x_k)\|\,\|s\| + \frac{L}{2}\|s\|^2 \leq K_3\|s\|$$

for a suitably defined constant K_3. If $k + 1 \not\equiv 0 \pmod{q}$, then by (27) and (28),

$$\|t_{k+1}\|_G \leq \|t_k\|_G + K_2 \frac{\|y\|}{\|s\|} \leq \|t_k\|_G + K_2 K_3. \tag{30}$$

Hence, by (30),

$$\begin{aligned} \|H_{k+1}\| = \|T_{k+1}\| &\leq \|T_{k+1}\|_F = \|t_{k+1}\|_G \leq \|t_k\|_G + K_2 K_3 \\ &= \|T_k\|_F + K_2 K_3 \leq \sqrt{n}\|T_k\| + K_2 K_3 = \sqrt{n}\|H_k\| + K_2 K_3 \\ &= (\sqrt{n})^q M + q\sqrt{n} K_2 K_3 . \end{aligned}$$

□

COROLLARY 3.2. *Under the hypothesis of Theorem* 3.7, *the sequence $\{x_k\}$ is well defined by Algorithms* 2.1 *and* 3.1, *and there is at least one limit point of the sequence. Every limit point is a stationary point for f.*

Proof. The proof follows directly from Theorem 2.4, Theorem 3.7, and the compactness of L_0. □

4. Local superlinear convergence. In §3, we proved that Algorithm 2.1, with the approximate Hessian matrices $\{H_k\}$ chosen by Algorithm 3.1, is globally convergent in the sense that every limit point of the sequence $\{x_k\}$ must satisfy the first-order stationary condition. In this section, we will do two things at once by doing a local analysis of the direct-prediction method associated with the tridiagonal factor update method. This means that we will take $x_{k+1} = x_k + s_k^N$. Unhappily, the good local

behavior of this iteration imposes that $H_k \equiv \nabla^2 f(x_k)$ if $k \equiv 0 \pmod{q}$. First, we will prove some strong bounded deterioration results for $\{H_k\}$ which will be crucial to our global convergence result in §5. Then, almost as a sidelight to the main theme of this paper, we will prove that the direct-prediction method is locally q-superlinearly convergent to stationary points at which the Hessian is nonsingular. It will turn out that this result is also useful in the global analysis of §5.

Let us define the algorithm under consideration in this section as an independent algorithm.

ALGORITHM 4.1. Assume that $x_0 \in \mathbb{R}^n$, $H_0 = \nabla^2 f(x_0)$. Given $x_k \in \mathbb{R}^n$, $H_k \in \mathbb{R}^{n\times n}$, $H_k = Q_k T_k Q_k^T$, Q_k orthogonal, $T_k \in \Upsilon$, obtain x_{k+1}, H_{k+1} as follows:

Step 1. $x_{k+1} = x_k - H_k^+ \nabla f(x_k)$.

Step 2. If $k+1 \equiv 0 \pmod{q}$, set $H_{k+1} = \nabla^2 f(x_{k+1})$. Else, obtain H_{k+1} using Algorithm 3.1.

Let us state the assumptions on f which allow us to obtain a local superlinear convergence result.

ASSUMPTION 4.1. Let $f \in C^2(\Omega)$, Ω an open and convex set. We assume that $x_* \in \Omega$ is such that $\nabla f(x_*)$ is symmetric and nonsingular. Further, we assume that (29) holds for all $x, w \in \Omega$.

Let P_Υ denote the Frobenius norm projection operator onto the subspace of symmetric tridiagonal matrices Υ.

LEMMA 4.1. *Assume that* $k \equiv 0 \pmod{q}$ *and that* $x_k \in \Omega$ *is well defined. Then,*

$$\|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F \le 2\sqrt{n}\, L \|x_k - x_*\| .$$

Proof.

$$\begin{aligned} &\|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \\ &\quad \le \|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_k) Q_k\|_F \\ &\qquad + \|{Q_k}^T \nabla^2 f(x_k) Q_k - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F . \end{aligned}$$

But ${Q_k}^T \nabla^2 f(x_k) Q_k \in \Upsilon$. Therefore,

$$\begin{aligned} &\|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_k) Q_k\|_F \\ &\qquad = \|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - P_\Upsilon({Q_k}^T \nabla^2 f(x_k) Q_k)\|_F \\ &\qquad \le \|{Q_k}^T \nabla^2 f(x_*) Q_k - {Q_k}^T \nabla^2 f(x_k) Q_k\|_F . \end{aligned}$$

Hence, by (29),

$$\begin{aligned} \|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F &\le 2\|Q_k^T(\nabla^2 f(x_k) - \nabla^2 f(x_*)) Q_k\|_F \\ &\le 2\sqrt{n}\, \|{Q_k}^T(\nabla^2 f(x_k) - \nabla^2 f(x_*)) Q_k\| \\ &= 2\sqrt{n}\, \|\nabla^2 f(x_k) - \nabla^2 f(x_*)\| \\ &= 2\sqrt{n}\, L\|x_k - x_*\|. \end{aligned}$$

□

From now on, let us use the notation $e_\ell = \|x_\ell - x_*\|$, $\ell = 0, 1, 2, \cdots$.

LEMMA 4.2. *Assume that* $k \equiv 0 \pmod{q}$, $0 \le j \le q-1$, *and that* x_{k+j}, x_{k+j+1}, $x_{k+j} + s_{k+j}$ *are well defined and belong to* Ω. *Then,*

$$\|y_{k+j} - [P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)] s_{k+j}\| \le L\|s_{k+j}\|(e_{k+j} + \tfrac{1}{2} e_{k+j+1} + 2\sqrt{n}\, e_k) .$$

Proof.

$$\|y_{k+j} - [P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)] s_{k+j}\| \le \|y_{k+j} - {Q_k}^T \nabla^2 f(x_*) Q_k s_{k+j}\| + \|P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_*) Q_k\| \, \|s_{k+j}\| \, . \tag{31}$$

But, by (29), and the definition of y_{k+j},

$$\begin{aligned} &\|y_{k+j} - {Q_k}^T \nabla^2 f(x_*) Q_k s_{k+j}\| \\ &\quad = \|g(x_{k+j} + Q_k s_{k+j}) - g(x_{k+j}) - \nabla^2 f(x_*) Q_k s_{k+j}\| \\ &\quad \le \frac{L}{2} \|s_{k+j}\| \max\{e_{k+j}, \|x_{k+j} + Q_k s_{k+j} - x_*\|\} \, . \end{aligned} \tag{32}$$

Therefore, by (31), (32), and Lemma 4.1,

$$\begin{aligned} &\|y_{k+j} - [P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)] s_{k+j}\| \\ &\quad \le \frac{L}{2} \|s_{k+j}\| \max\{e_{k+j}, \|x_{k+j} + Q_k s_{k+j} - x_*\|\} + 2\sqrt{n}\, L \|s_{k+j}\| e_k \, . \end{aligned}$$

Now, even if $s_{k+j} \neq x_{k+j+1} - x_{k+j}$, they are equal in norm, so

$$\begin{aligned} \|x_{k+j} - Q_k s_{k+j} - x_*\| \le e_{k+j} + \|s_{k+j}\| &= e_{k+j} + \|x_{k+j+1} - x_{k+j}\| \\ &\le 2e_{k+j} + e_{k+j+1} \, . \end{aligned}$$

Therefore,

$$\begin{aligned} &\|y_{k+j} - [P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)] s_{k+j}\| \\ &\quad \le \frac{L}{2} \|s_{k+j}\| (2e_{k+j} + e_{k+j+1}) + 2\sqrt{n}\, L \|s_{k+j}\| e_k \\ &\quad \le L \|s_{k+j}\| (e_{k+j} + \tfrac{1}{2} e_{k+j+1} + 2\sqrt{n}\, e_k) \, , \end{aligned}$$

as we wanted to prove. □

The following lemma states a Bounded Deterioration Principle (see [1]) for the matrices T_k.

LEMMA 4.3. *Assume that* $k \equiv 0 \pmod q$, $0 \le j \le q-2$, *and that* x_{k+j}, x_{k+j+1}, $x_{k+j} + Q_k s_{k+j}$ *are well defined and belong to* Ω. *Then,*

$$\begin{aligned} &\|T_{k+j+1} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)\|_F \\ &\quad \le \|T_{k+j} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)\|_F + K_2 L (e_{k+j} + \tfrac{1}{2} e_{k+j+1} + 2\sqrt{n}\, e_k) \, . \end{aligned}$$

Proof. For matrices $T \in \Upsilon$, remember that $\|T\|_F = \|\Phi(T)\|_G$, where Φ is the isomorphism which maps Υ into $\mathbb{R}^{2n-1}$. The matrices

$$T_{k+j+1} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)$$

and

$$T_{k+j} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)$$

belong to Υ. So, using the convention $t = \Phi(T)$, we are going to prove the thesis in $\mathbb{R}^{2n-1}$ using $\|\cdot\|_G$.

By (26) we have, writing $y = y_{k+j}$,

$$t_* = \Phi(P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k)),$$
$$t_{k+j+1} = t_{k+j} - G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} (A_{k+j} t_{k+j} - y).$$

So,

$$\begin{aligned} t_{k+j+1} - t_* &= t_{k+j} - t_* - G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} (A_{k+j} t_{k+j} - y) \\ &= t_{k+j} - t_* - G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} \\ &\quad \times (A_{k+j} t_{k+j} - A_{k+j} t_* + A_{k+j} t_* - y) \\ &= [I - G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} A_{k+j}](t_{k+j} - t_*) \\ &\quad + G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} (y - A_{k+j} t_*). \end{aligned}$$

Hence,

$$\begin{aligned} \| t_{k+j+1} - t_* \|_G &\le \| [I - G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} A_{k+j}](t_{k+j} - t_*) \|_G \\ &\quad + \| G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} (y - A_{k+j} t_*) \|_G \\ &\le \| t_{k+j} - t_* \|_G + \| G^{-1} A_{k+j}^T (A_{k+j} G^{-1} A_{k+j}^T)^{-1} (y - A_{k+j} t_*) \|_G . \end{aligned}$$

Therefore, using the arguments which lead to (28), we have:

$$\| t_{k+j+1} - t_* \|_G \le \| t_{k+j} - t_* \|_G + \frac{K_2 \| y - A_{k+j} t_* \|}{\| s_{k+j} \|} .$$

But $A_{k+j} t_* = P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) s_{k+j}$. Thus, the desired result follows using Lemma 4.2. □

LEMMA 4.4. *Assume that* $k \equiv 0 \pmod q$, $0 \le j \le q-2$, *and that* x_{k+j}, x_{k+j+1}, $x_{k+j} + Q_k s_{k+j}$ *are well defined and belong to* Ω. *Then,*

$$\begin{aligned} &\| T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \\ &\quad \le \| T_{k+j} - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F + L \left(K_2 e_{k+j} + \frac{K_2}{2} e_{k+j+1} + 2\sqrt{n}\,(K_2 + 1) e_k \right) . \end{aligned}$$

Proof. By Lemmas 4.1 and 4.3, we have:

$$\begin{aligned} &\| T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \\ &\quad \le \| T_{k+j+1} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) \|_F \\ &\qquad + \| P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \\ &\quad \le \| T_{k+j} - P_\Upsilon({Q_k}^T \nabla^2 f(x_*) Q_k) \|_F \\ &\qquad + K_2 L (e_{k+j} + \tfrac{1}{2} e_{k+j+1} + 2\sqrt{n}\, e_k) + 2 L e_k \sqrt{n} \end{aligned}$$

and the desired result follows trivially from this inequality. □

LEMMA 4.5. *Assume the hypotheses of the previous lemmas. Then,*

$$\| T_k - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \le \sqrt{n} L e_k , \tag{33}$$

and for $0 \le j \le q-2$,

$$\begin{aligned} &\| T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k \|_F \\ &\quad \le \sqrt{n}\, L e_k + \sum_{\nu=0}^{j} L \left(K_2 e_{k+\nu} + \frac{K_2}{2} e_{k+\nu+1} + 2\sqrt{n}\,(K_2 + 1) e_k \right) . \end{aligned}$$

Proof.

$$\|T_k - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F = \|{Q_k}^T \nabla^2 f(x_{k+j}) Q_k - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F$$
$$\leq \sqrt{n}\, \|\nabla^2 f(x_{k+j}) - \nabla^2 f(x_*)\| \leq \sqrt{n}\, L e_{k+j} = \sqrt{n}\, L e_k\ .$$

Thus the desired result follows straightforwardly from the previous inequality and Lemma 4.4. □

LEMMA 4.6. *Assume the hypotheses of the previous lemmas, and remember that*

$$H_\ell = Q_\ell T_\ell Q_\ell^T \qquad \text{for } \ell = 1, 2, \cdots.$$

Then, for some $\eta \geq 0$

$$\|H_{k+j+1} - \nabla^2 f(x_*)\| \leq \eta \left(\sum_{\nu=0}^{j+1} e_{k+\nu} \right).$$

Proof. By Lemma 4.5,

$$\begin{aligned} \|H_{k+j+1} - \nabla^2 f(x_*)\| &= \|Q_k (T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k) {Q_k}^T\| \\ &\leq \|T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k\| \\ &\leq \|T_{k+j+1} - {Q_k}^T \nabla^2 f(x_*) Q_k\|_F \\ &\leq \sqrt{n}\, L e_k + \sum_{\nu=0}^{j} L \left(K_2 e_{k+\nu} + \frac{K_2}{2} e_{k+\nu+1} + 2\sqrt{n}\,(K_2+1) e_k \right) \end{aligned}$$

and the result follows directly. □

THEOREM 4.7. *There exists* $\epsilon > 0$ *such that for any* x_0 *with* $\|x_0 - x_*\| \leq \epsilon$, *the sequence* $\{x_\ell\}$ *generated by Algorithm* 4.1 *converges q-superlinearly to* x_*. *Furthermore, if* $\epsilon q \eta \|\nabla^2 f_*^{-1}\| \leq \gamma < 1$, *then the sequence* $\{\|H_\ell^{-1}\|\}$ *is uniformly bounded by the constant*

$$B_N \equiv \frac{\|\nabla^2 f(x_*)^{-1}\|}{(1-\gamma)}$$

independent of the particular choice of x_0.

Proof. Algorithm 4.1 is locally linearly convergent and $\{\|H_\ell^{-1}\|\}$ is uniformly bounded if the matrices H_k remain in a suitable neighborhood of $\nabla^2 f(x_*)$. (See [3, Chap. 7].) This condition is easily verified using Lemma 4.6 if x_0 is close enough to x_*. The reason this condition and the bound on the inverses can be independent of the particular x_0 is that Algorithm 4.1 always takes $H_0 = \nabla^2 f(x_0)$. In particular,

$$\|H_\ell - \nabla^2 f(x_*)\| \leq \epsilon q \eta$$

and so the bound B_N follows from the Banach lemma (see [3]). Now, using linear convergence and Lemma 4.6, we see that $\lim_{k\to\infty} H_k = \nabla^2 f(x_*)$. This implies that convergence is q-superlinear (see [1]). □

5. Global superlinear convergence. In §3, we proved that Algorithm 2.1, with the approximate Hessian matrices $\{H_k\}$ chosen by Algorithm 3.1, is globally convergent in the sense that every limit point of the sequence $\{x_k\}$ is a first-order stationary point. In §4, we proved that if we require the Hessian update method to always choose $H_k = \nabla^2 f(x_k)$ every q iterations, then the direct-prediction method is locally q-superlinearly convergent to stationary points at which the Hessian is nonsingular. In this section, we put all this together. We update the Hessian approximations as in §4, and we modify Algorithm 2.1 to always try the full quasi-Newton step first when H_k is positive definite. We then prove that if f is quasi-convex on L_0 and $\nabla^2 f(x_*) = \nabla^2 f(x_*)$ is positive definite for some stationary point x_*, then from some point on, the Newton steps satisfy the sufficient decrease condition (2).

ALGORITHM 5.1 Assume that $x_0 \in \mathbb{R}^n$, $H_0 = \nabla^2 f(x_0)$. Given $x_k \in \mathbb{R}^n$, $H_k \in \mathbb{R}^{n\times n}$, $H_k = Q_k T_k Q_k^T$, Q_k orthogonal, $T_k \in \Upsilon$, obtain $\{x_{k+1}\}$, $\{H_{k+1}\}$ as follows:

Step 1. If H_k is positive definite, then in Algorithm 2.1, first try $x_{k+1} = x_k - H_k^{-1}\nabla f(x_k)$.

Step 2. If $k+1 \equiv 0 \pmod q$, set $H_{k+1} = \nabla^2 f(x_{k+1})$. Else, obtain H_{k+1} using Algorithm 3.1. Return to Step 1.

Now we give our main result. We assume that f is quasi-convex, i.e., that all level sets of f are convex.

THEOREM 5.1. *Let $f \in C^2(\Omega)$, Ω an open and convex set containing L_0, be a quasi-convex function on L_0. Assume that L_0 is bounded, and that some stationary point $x_* \in \Omega$ is such that $\nabla^2 f(x_*)$ is positive definite. Further, assume that the Lipschitz condition on the Hessian given by* (29) *holds for all $x, w \in \Omega$. Then, there exists some integer k_N such that Algorithm* 5.1 *takes $\mu_k = 0$ for $k \geq k_N$, and $\{x_k\}$ converges q-superlinearly to x_*, which is the global minimizer of f.*

Proof. Since f is quasi-convex and has a stationary point x_* at which $\nabla^2 f(x_*)$ is positive definite, x_* must be the unique stationary point for f on L_0, and the global minimizer of f.

Since L_0 is bounded and $\nabla^2 f$ is continuous, we can take

$$\mathcal{H} = \{\nabla^2 f(x) : x \in L_0\}.$$

Thus from Corollary 3.2, we have that $\{x_k\}$ is well defined and that some subsequence converges to a stationary point, which must then be x_*. Furthermore, there is some $B \geq \|H_k\|$ uniformly in k. Since x_* is the only possible limit point of $\{x_k\}$, the compactness of L_0 ensures that $\lim_k x_k = x_*$. In particular, the subsequence of the iterates indexed by $k \equiv 0 \pmod q$ converges to x_*.

The key to the proof will be to show below that eventually, starting at one of the $k \equiv 0 (\text{mod } q)$ iterates, Algorithm 5.1 reduces to Algorithm 4.1, i.e., the step $s_k^N = -H_k^{-1} g_k$ eventually satisfies (2).

Let ϵ be small enough that Algorithm 4.1 is locally q-linearly convergent to x_* from any x_0^N with $\|x_0^N - x_*\| < \epsilon$. Now, let B_N be as in Theorem 4.7. Choose ϵ even smaller, if necessary, to make $1 - 2\alpha > (q\eta + L)B_N\epsilon$. The standard approach to proving Theorem 4.7 makes ϵ be chosen so that if $\nabla^2 f(x_*)$ is positive definite, then so are all H_k^N for $\|x_0^N - x_*\| < \epsilon$. Choose $k_N \equiv 0 \pmod q$ so that if $k \geq k_N$, then $\|x_k - x_*\| < \epsilon$.

There are still a couple of small points to deal with before we start to chain inequalities. First, since H_k is positive definite, we have $g_k^T s_k^N < 0$, and

$$\|s_k^N\|^2 = (H_k^{-1/2} g_k)^T H_k^{-1} H_k^{-1/2} g_k \leq \|(H_k)^{-1}\| (H_k^{-1/2} g_k)^T H_k^{-1/2} g_k$$

$$\leq -\|(H_k)^{-1}\| g_k^T s_k^N .$$

Furthermore, $x_k^N, x_k^N + s_k^N$ are both within ϵ of x_*. Thus, any convex combination is also, and so for any $\xi \in (0,1)$, $\|x_k^N + \xi s_k^N - x_*\| < \epsilon$.

Now the proof that $\{x_k^N\} = \{x_k\}$ for $k > k_N$ is by Taylor's theorem and all these partial results. It can be done by induction, but we give only the main step here. Assume that the sequences are identical from the k_Nth to the ℓth iterate. Then $H_\ell = H_{\ell - k_N}^N$, and

$$\begin{aligned} f(x_\ell + s_\ell^N) - f_\ell &= g_\ell^T s_\ell^N + \tfrac{1}{2}(s_\ell^N)^T[\nabla^2 f(x_\ell + \xi s_\ell^N - x_*) \pm \nabla^2 f(x_*)] s_\ell^N \\ &= \tfrac{1}{2} g_\ell^T s_\ell^N + \tfrac{1}{2}(s_\ell^N)^T[\nabla^2 f(x_\ell + \xi s_\ell^N - x_*) \pm \nabla^2 f(x_*) - H_\ell] s_\ell^N \\ &\leq \tfrac{1}{2} g_\ell^T s_\ell^N + \tfrac{1}{2}[L + q\eta]\epsilon \|s_\ell^N\|^2 \\ &\leq \tfrac{1}{2} g_\ell^T s_\ell^N - \tfrac{1}{2}[L + q\eta]\epsilon B_N g_\ell^T s_\ell^N \\ &\leq \tfrac{1}{2} g_\ell^T s_\ell^N - \tfrac{1}{2}(1 - 2\alpha) g_\ell^T s_\ell^N = \alpha g_\ell^T s_\ell^N \end{aligned}$$

since H_ℓ is positive definite and so $g_\ell^T s_\ell^N < 0$. □

6. Implementation.

6.1. Implementation of steps 4 and 5 of Algorithm 2.1. Considering $s_k(\mu) = -(H_k + \mu I)^{-1} g(x_k)$ with

$$\mu \geq \bar{\mu} = \max(0, -\lambda_1 + \epsilon),$$

where λ_1 is the least eigenvalue of H_k and $\epsilon = 10^{-5}$ in the computer implementation, we choose

$$x_{k+1} = x_k + s_k(\mu_*),$$

where μ_* is an approximate solution to the problem

$$\text{(I)} \qquad \arg\min f(x_k + s_k(\mu)), \qquad \mu \geq \bar{\mu} .$$

In order to solve this problem it is necessary to follow the curvilinear path $s_k(\mu)$, $\mu \geq \bar{\mu}$, and therefore to find the solution of the linear system of equations

$$(H_k + \mu I) s_k(\mu) = -g(x_k) , \qquad \mu \geq \bar{\mu}$$

for several trial values of μ. These computations are carried out in $O(n)$ operations because the decomposition $H_k = Q_k T_k Q_k{}^T$ is available. This is because we can write the equivalent system

$$(T_k + \mu I)\hat{s}_k(\mu) = -\hat{g}(x_k),$$

where $\hat{s}_k(\mu) = Q_k{}^T s_k(\mu)$, $\hat{g}(x_k) = Q_k{}^T g(x_k)$.

The least eigenvalue of T_k is obtained by means of the IMSL routine EQRT1S, and the solution of the tridiagonal systems by the LINPACK routine SGTSL.

For solving (I) we modified the routine GSRCH originally written by Powell for MINPACK [10].

The new iterate x_{k+1} is accepted (step 5 of Algorithm 2.1) only if the condition

$$f(x_{k+1}) \leq f(x_k) + \alpha g(x_k)^T (x_{k+1} - x_k)$$

is satisfied with $\alpha = 10^{-4}$. However, we may continue searching even if the Newton step satisfies this criterion.

We decide that $\Gamma_2(x_k, H_k)$ is not empty if the angle between g_k and $v_1^{(k)}$ is between 85 degrees and 95 degrees.

6.2. Choosing the sequence B_k. For those iterations in which $H_k = \nabla^2 f(x_k)$, the decomposition is computed with the IMSL routines EHOUSS and EHOBS, except when the Hessian itself is tridiagonal.

The stopping condition is (7.2.5) of Dennis and Schnabel [3, p. 160]

$$\max_{1\le i\le n}\left\{\frac{|\nabla f(x_k)_i|\max(|x_i^k|,1)}{\max(|f(x_k)|,1)}\right\} \le \text{eps}$$

(eps $= 10^{-15}$ in the computer implementation).

6.3. Efficiency. The computer program allows the user to compute the full decomposition every q iterations (we use $q = 3$) or to decide when to do so in between automatically, depending upon the following notion of efficiency of an iteration. We define efficiency of the kth iteration as

$$E_k = \frac{-\log r_k}{t_k},$$

where

$$r_k = \frac{f_{k+1} - f_*}{f_k - f_*},$$

f_* is an estimation of $f(x_*)$, $f_{k+1} = f(x_{k+1})$, and t_k is the CPU time required by the kth iteration.

Assuming r_k remains constant until convergence (denoted by r hereafter), the required number of iterations NITER is approximately given by

$$r^{NITER} = \text{eps}\,.$$

Therefore, the total CPU time T will be

$$T = \frac{\log \text{eps}}{\log r} t_k = -\frac{\log \text{eps}}{E_k}\,.$$

In order to decide what H_{k+1} will be (that is, $H_{k+1} = \nabla^2 f(x_{k+1})$ or $H_{k+1} = Q_k T_{k+1} Q_k{}^T$), we use E_k as follows. Let k_0 be the last iteration such that $B_{k_0} = \nabla^2 f(x_{k_0})$. If $k_0 \equiv k \pmod q$ or if $E_{k_0} > E_k$, then $H_{k+1} = \nabla^2 f(x_{k+1})$. Otherwise $H_{k+1} = Q_k T_{k+1} Q_k{}^T$.

7. Numerical experience. The class of algorithms described in the previous sections form the theoretical basis of subroutine TRIDI.

The decision about when Γ_2 is not empty is taken according to a user-supplied parameter defining a maximum deviation in degrees with respect to orthogonality. This parameter was defined as five degrees for the numerical experiments.

7.1. Test problems. In order to demonstrate the effectiveness of the new method, numerical results were obtained not only for well-known test examples appearing in the literature but also for some new functions. For brevity, the full details of the test problems are not given here except for the following new ones:

TEST FUNCTION PRUEBA.

$$f(x) = a(1)/x(1) + a(2)/x(2) + a(3)/x(3) + 0.5\langle x, Cx\rangle + \langle b, x\rangle$$

where $b(i) = 1. \times 10^{-6} * a(i) - (i+4) * 1. \times 10^3$ for $i = 1, \cdots, 3$, a is as defined in Table 1, and

$$C = \begin{pmatrix} 1/3 & 1/10 & 1/10 \\ 1/10 & 1/4 & 1/10 \\ 1/10 & 1/10 & 1/5 \end{pmatrix}.$$

The underlying idea is that if a starting point is close to the origin, the "wavy behavior" of the function leads to a very small trust region, a phenomenon which leads to a rather inefficient performance of the classical method. This shortcoming does not exist for the new algorithm because of the curvilinear search, which can be considered as a way of computing an optimal radius in each iteration.

TEST FUNCTION SNLLSQ I. Generate data $(j, y(j))$ for $j = 1, \cdots, 15$ from

$$y(j) = a(1) * j**xopt(1) + a(2) * j**xopt(2) + a(3) * j**xopt(3)$$

with $a(1) = 3$, $a(2) = 3.1$, $a(3) = 0.7$, $xopt(1) = 1.5$, $xopt(2) = 2.5$, $xopt(3) = -2.5$.

Now with the given a, recover x by a least-squares fit to this data.

TEST FUNCTION SNLLSQ II. Generate data $(j, y(j))$ for $j = 1, \cdots, 15$ from

$$y(j) = a(1) * \sin(j * xopt(1)) + a(2) * \sin(j * xopt(2)) + a(3) * \sin(j * xopt(3))$$

with $a(i), xopt(i)$, $i = 1, \cdots, 3$ as in SNLLSQ I. Again, recover x by least squares.

TEST FUNCTION SNLLSQ III. Generate data $(j, y(j))$ for $j = 1, \cdots, 30$ from

$$y(j) = a(1) * \cos(j * xopt(1)) + a(2) * \cos(j * xopt(2)) + a(3) * \cos(j * xopt(3))$$

with $a(1) = 10$, $a(2) = 20$, $a(3) = 30$, $xopt(1) = 0.1$, $xopt(2) = 0.2$, $xopt(3) = 0.3$.

Recover x by least squares.

TEST FUNCTION SNLLSQ IV. Generate data $(j, y(j))$ for $j = 1, \cdots, 45$ from

$$y(j) = a(1) * \exp(j * xopt(1)) + a(2) * \exp(j * xopt(2)) + a(3) * \exp(j * xopt(3))$$

with $a(1) = 1$, $a(2) = 2$, $a(3) = 3$, $xopt(1) = -0.1$, $xopt(2) = -0.2$, $xopt(3) = -0.3$.

Now recover x by least squares.

From here on we use the notation tfn.n.cn.sp, where tfn is the test function number, n the number of variables, cn the case number, and sp the identification of the starting point.

Table 1 defines the problems.

TABLE 1

tfn	Name	n	cn	sp
1	Prueba	3	1: $a(i) = 1.d - 1$	1: $(1.d - 3,\ 1.d - 3,\ 1.d - 3)$
1		3	2: $a(1) = 1.d3$ $a(2) = a(3) = 1.d0$	2: (0.25, 0.25, 0.25)
1		3	3: $a(1) = a(2) = a(3) = 1.d1$	
2	Penalty I	4	1	1: $x(j) = j$
2	[3]	8	1	
3	Variable Dimensioned	4	1	1: $x(j) = 1 - j/n$
3	[3]	5		
3		8		
3		12		

TABLE 1
(*continued*)

tfn	Name	n	cn	sp
4	Rosenbrock	4	1	1: $x(2j-1) = -1.2,\ x(2j) = 1$
4	[3]	8		
4		10		
4		12		
5	Chained Rosenbrock [3]	25	1	1: $x(j) = -1$
6	Powell Extended	4	1	1: $x(4j-3) = 3,\ x(4j-2) = -1$
6	[3]	8	1	$x(4j-1) = 0,\ x(4j) = 1$
6		240	1	
6		400	1	
7	Brown–Dennis	4	1	1: $(25, 5, -5, 1)$
8	Gaussian [3]	3	1	1: $(0.4, 1, 0)$
9	Trigonometric	25	1	1: $x(j) = 1$
9	[5]	50		
9		100		
9		200		
10	Watson [3]	12	1	1: $x(j) = 0$
11	Wood [3]	4	1	1: $(-3, -1, -3, -1)$
12	Box [3]	3	1	1: $(0, 10, 20)$
13	Biggs Exp 6 [3]	6	1	1: $(1.2, 1, 1, 1, 1, 1)$
14	Dennis–Marwil I [2]	10	1: $r1 = 1;\ r2 = n$ $k1 = k3 = 1;\ k2 = 5$ 2: $r1 = 1;\ r2 = n$ $k1 = 4;\ k2 = k3 = 1$	1: $x(j) = -1$
		100	2	
15	Dennis–Marwil II [2]	5	1	1: $x(j) = -1$
16	Pseudo Penalty [3]	50	1	1: $x(j) = 0$
17	SNLLSQ I	3	1	1: $x(j) = 3.50 * xopt(j)$
18	SNLLSQ II	3	1	1: $x(j) = 1.15 * xopt(j)$
19	SNLLSQ III	3	1	1: $x(j) = 1.50 * xopt(j)$
20	SNLLSQ IV	3	1	1: $x(j) = 3.00 * xopt(j)$

TABLE 2

Problem	NIT	FE	GE	HE	T	FMIN
1.3.1.1	18	33	19	18	1.00	$-.13e+09$
	11	12	12	5	0.29	$-.13e+09$
	13	14	14	4	0.33	$-.13e+09$
1.3.1.2	12	14	13	12	1.00	$-.13e+09$
	5	6	6	2	0.24	$-.13e+09$
	5	6	6	2	0.22	$-.13e+09$
1.3.2.1			error 6			
	23	24	24	9	0.09	$-.13e+09$
	22	23	23	8	0.06	$-.13e+09$
1.3.2.2	13	26	14	13	1.00	$-.13e+09$
	8	9	9	3	0.26	$-.13e+09$
	8	9	9	2	0.24	$-.13e+09$
1.3.3.1	23	33	24	23	1.00	$-.13e+09$
	17	18	18	8	0.45	$-.13e+09$
	18	19	19	5	0.45	$-.13e+09$
1.3.3.2	12	20	13	12	1.00	$-.13e+09$
	5	6	6	2	0.20	$-.13e+09$
	5	6	6	2	0.19	$-.13e+09$
2.4.1.1	34	48	35	34	1.00	$0.23e-04$
	11	12	12	5	0.50	$0.24e-04$
	12	13	13	4	0.33	$0.24e-04$
2.8.1.1	34	43	35	34	1.00	$0.54e-04$
	15	16	16	5	0.88	$0.57e-04$
	17	21	21	6	1.09	$0.57e-04$
3.4.1.1	10	11	11	10	1.00	$0.24e-27$
	12	13	13	5	1.10	$0.21e-30$
	12	13	13	4	1.88	$0.78e-12$
3.5.1.1	11	12	12	11	1.00	$0.13e-28$
	14	15	15	6	3.79	$0.27e-19$
	14	34	34	4	3.74	$0.61e-17$

7.2. Numerical results. Table 2 gives the obtained numerical results using the notation:
NIT = number of iterations
FE = number of function evaluations
GE = number of gradient evaluations
HE = number of Hessian evaluations
T = relative CPU time with respect to the IMSL routines
FMIN = Computed minimum

For each problem three sets of results are given; the first row corresponds to the routine DUMIAH (trust region algorithm), the second and third to the new method with efficiency and without efficiency, respectively. For the last four test problems the first row corresponds to the results obtained with the routine DUMIDH. Error 6 in DUMIAH means that five consecutive steps have been taken with the maximum step length.

The computational tests were carried out in double precision on a Hewlett-Packard 9000 825S computer using software written in Fortran 77 under the HP-UX operating system and on an IBM 4361. The reason for using two different computers

TABLE 2
(*continued*)

Problem	NIT	FE	GE	HE	T	FMIN
3.8.1.1	13	14	14	13	1.00	$0.53e-26$
	17	18	18	5	4.75	$0.22e-24$
	16	18	18	6	4.75	$0.19e-16$
3.10.1.1	14	15	15	14	1.00	$0.18e-25$
	18	21	21	5	7.07	$0.15e-14$
	18	19	19	6	6.13	$0.46e-19$
4.4.1.1	23	34	24	23	1.00	$0.55e-20$
	31	50	49	14	1.16	$0.39e-31$
	39	72	71	10	1.45	$0.77e-21$
4.8.1.1	23	34	24	23	1.00	$0.11e-19$
	35	63	61	16	1.85	$0.29e-27$
	42	91	88	11	2.21	$0.34e-23$
4.10.1.1	23	34	24	23	1.00	$0.14e-19$
	36	75	73	12	1.68	$0.28e-11$
	36	75	73	12	1.46	$0.23e-11$
4.12.1.1	23	34	24	23	1.00	$0.16e-19$
	38	87	84	13	1.80	$0.18e-15$
	38	87	84	13	1.78	$0.18e-15$
5.25.1.1	15	19	16	15	1.00	$0.14e-13$
	19	51	49	7	0.62	$0.13e-15$
	19	51	49	7	0.56	$0.13e-15$
6.4.1.1	15	17	16	15	1.00	$0.46e-08$
	19	20	20	7	1.10	$0.46e-08$
	19	20	20	7	1.00	$0.47e-08$
6.8.1.1	15	17	16	15	1.00	$0.92e-08$
	22	27	27	8	1.58	$0.63e-08$
	22	27	27	8	1.68	$0.63e-08$
6.240.1.1	15	17	16	15	1.00	$0.27e-06$
	23	38	39	6	0.39	$0.93e-06$
	20	39	40	7	0.47	$0.19e-05$
6.400.1.1	15	17	16	15	1.00	$0.45e-06$
	23	36	37	6	0.33	$0.16e-05$

was mainly that the efficiency idea is quite sensitive to the precision with which the CPU time is measured. Due to the fact that timing routines like the one provided in the IMSL Library or others available for UNIX systems do not fulfill the accuracy requirements in the sense that different runs of the same problem may give unacceptable differences for our purposes, some of the small-size problems were run on an IBM computer for which the staff of the University of LaPlata Computer Center wrote a very precise assembler routine for measuring CPU time. For several reasons, it was not feasible to run all examples on that computer, so most of the results are from the HP machine. In order to normalize comparisons, all results are given relative to the CPU time required by the IMSL optimization routines except in the examples in which they failed to converge properly. All comparisons of the new method have been made against the trust regions algorithm as implemented in subroutine DUMIAH of the IMSL Library (version 1.0, April 1987), with the only exception being the separable nonlinear least squares problems for which subroutine DUMIDH was

TABLE 2
(*continued*)

Problem	NIT	FE	GE	HE	T	FMIN
7.4.1.1	8	10	9	8	1.00	$0.86e+05$
	9	16	16	5	1.26	$0.86e+05$
	13	19	19	4	1.39	$0.86e+05$
8.3.1.1	1	4	2	1	1.00	$0.11e-07$
	2	3	3	1	0.41	$0.11e-07$
	2	3	3	1	0.47	$0.11e-07$
9.25.1.1	6	20	7	6	1.00	$-0.75e+04$
	9	22	22	3	0.94	$-0.75e+04$
	9	22	22	3	0.94	$-0.75e+04$
9.50.1.1	8	26	9	8	1.00	$-0.31e+05$
	13	16	15	6	0.90	$-0.31e+05$
	17	28	27	5	0.91	$-0.31e+05$
9.100.1.1	17	39	18	17	1.00	$-0.12e+06$
	20	45	45	7	0.68	$-0.12e+06$
	20	45	45	7	0.58	$-0.12e+06$
9.200.1.1	23	43	64	35	1.00	$-0.50e+06$
	22	43	43	8	0.70	$-0.50e+06$
	22	43	43	8	0.72	$-0.50e+06$
10.12.1.1	12	26	13	12	1.00	$0.22e-07$
	22	52	48	8	0.97	$0.23e-07$
	22	52	48	8	0.85	$0.22e-07$
11.4.1.1	12	26	13	12	1.00	$0.47e-09$
	12	59	56	7	0.76	$0.49e-07$
	12	61	57	8	1.03	$0.15e-07$
12.3.1.1	7	14	8	7	1.00	$0.54e-16$
	10	14	14	4	1.00	$0.14e-11$
	10	14	14	4	0.94	$0.14e-11$
13.6.1.1	29	60	30	29	1.00	$0.11e-11$
	33	52	46	13	0.77	$0.13e-12$
	53	85	77	14	1.19	$0.36e-12$
14.10.1.1	12	23	13	12	1.00	$0.29e-15$
	1	7	6	1	0.76	$0.23e-21$
	1	7	6	1	0.76	$0.23e-21$
14.100.2.1	17	37	18	17	1.00	$0.81e-15$
	1	6	6	1	0.16	$0.71e-25$
	1	6	6	1	0.16	$0.71e-25$
15.10.2.1	12	23	13	12	0.52	$0.17e-15$
	1	10	10	1	0.19	$0.38e-22$
	1	10	10	1	0.15	$0.38e-22$
15.5.1.1	4	6	5	4	1.00	$0.24e-13$
	5	6	6	2	1.00	$0.67e-12$
	5	6	6	2	1.01	$0.67e-12$
16.50.1.1	100	111	101	100	1.00	$0.23e-03$
	27	73	70	8	0.20	$0.23e-03$
	35	87	86	9	0.20	$0.23e-03$

TABLE 2
(*continued*)

Problem	NIT	FE	GE	HE	T	FMIN
17.3.1.1	7	78	29	0	1.73	$0.70e+02$
	64	182	237	0	5.91	$0.33e-18$
	56	140	194	0	4.45	$0.35e-12$
18.3.1.1			divergence			
	13	35	46	0	0.72	$0.15e-21$
	16	36	54	0	0.87	$0.92e-25$
19.3.1.1	26	84	105	0	2.97	$0.42e-18$
	31	37	64	0	1.61	$0.33e-18$
	31	39	72	0	1.54	$0.42e-22$
20.3.1.1	4	19	17	0	0.92	$0.17e-01$
	31	59	90	0	3.08	$0.33e-09$
	29	59	88	0	2.93	$0.93e-07$

used because a finite-difference Hessian was required.

In the following nonlinear least squares problems the absolute CPU time is given because of the poor performance of the trust-region algorithm, which led to divergence in one example, a large number of function evaluations in another, and to a very high functional value in the third.

The test examples show the new algorithm to be more robust (in fact, no example of divergence has been found) than the trust-region method, and that its efficiency tends to increase with the number of variables. This is so because of the savings in Hessian evaluations, and in spite of the CPU time spent on the computation of the least eigenvalue of the tridiagonal factor, which is relatively more important in small size problems.

7.3. Comparisons with not updating. In Table 3 are some examples to show that our update is better than if we kept the Hessian constant for q iterations. In particular, we compare not updating (we will call this method HC) against the method obtained updating the Hessian but without the test of §6.3 (WE = without efficiency).

The results of these tests convince us that our updating scheme is worthwhile. This is true despite the fact that no stronger convergence result holds for our updating scheme than for not updating.

Acknowledgments. The authors wish to thank Ms. Laura Carcione for programming help and for generating the test results.

TABLE 3

Problem	NIT	FE	GE	HE	T	FMIN	q	Method
1.3.2.1	22	23	23	8	1.00	$-0.13e+9$	4	WE
	40	41	41	14	1.39	$-0.13e+9$	4	HC
	21	22	22	4	1.00	$-0.13e+9$	6	WE
	63	64	64	11	1.73	$-0.13e+9$	6	HC
	31	32	32	4	1.00	$-0.13e+9$	10	WE
	91	92	92	10	1.62	$-0.13e+9$	10	HC
2.8.1.1	17	21	21	6	1.00	$+0.57e-4$	4	WE
	21	22	21	7	1.20	$+0.57e-4$	4	HC
	15	33	32	3	1.00	$+0.57e-4$	6	WE
	31	35	34	6	1.40	$+0.57e-4$	6	HC
	16	33	32	2	1.00	$+0.57e-4$	10	WE
	41	43	42	5	1.51	$+0.57e-4$	10	HC
10.12.1.1	22	52	48	8	1.00	$+0.22e-7$	4	WE
	51	61	58	17	2.31	$+0.43e-7$	4	HC
	37	98	92	7	1.00	$+0.24e-7$	6	WE
	72	177	159	12	1.66	$+0.42e-7$	6	HC
	51	108	102	6	1.00	$+0.43e-7$	10	WE
	96	260	232	10	1.69	$+0.43e-7$	10	HC
16.50.1.1	35	87	86	9	1.00	$+0.23e+3$	4	WE
	30	127	124	8	0.88	$+0.23e+3$	4	HC
	31	66	63	6	1.00	$+0.23e+3$	6	WE
	51	184	183	9	1.47	$+0.23e+3$	6	HC
	25	52	49	3	1.00	$+0.23e+3$	10	HC
	49	162	161	5	1.47	$+0.23e+3$	10	HC

REFERENCES

[1] C. G. BROYDEN, J. E. DENNIS, JR., AND J. J. MORÉ, *On the local and superlinear convergence of quasi-Newton methods*, IMA J. Appl. Math., 12 (1973), pp. 223–246.

[2] J. E. DENNIS, JR. AND E. S. MARWIL, *Direct secant updates of matrix factorizations*, Math. Comp., 38 (1982), pp. 459–474.

[3] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983. Russian edition: Mir Publishing Office, Moscow, 1988, O. Burdakov, trans.

[4] D. M. GAY, *Computing optimal locally constrained steps*, SIAM J. Sci. Statist. Comput., 2 (1981), pp. 186–197.

[5] J. M. MARTÍNEZ, *A new family of quasi-Newton methods for nonlinear equations with direct secant updates of matrix factorizations*, SIAM J. Numer. Anal., 27 (1990) pp. 1034–1049.

[6] ———, *On the order of convergence of the Broyden–Gay–Schnabel method*, Comm. Math. Univ. Carol, 19 (1978), pp. 107–118.

[7] ———, *A quasi-Newton method with a new updating for the* LDU *factorization of the approximate Jacobian*, Mat. Apl. Comput., 2 (1983), pp. 131–142.

[8] J. J. MORÉ AND D. C. SORENSEN, *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 553–572.

[9] A.M. OSTROWSKI, *Solution of Equations in Euclidean and Banach Spaces*, Academic Press, New York, 1973.

[10] M. J. D. POWELL AND Y. YUAN, *A trust region algorithm for equality constrained optimization*, DAMTP Report 1986/NA2, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, U.K., 1986.

[11] H. SCHRAMM AND J. ZOWE, *A combination of the bundle approach and the trust region concept*, Tech. Report Math. Inst. 1987/20, Mathematisches Institut, University of Bayreuth, Bayreuth, Germany, 1987.

[12] J.H. WILKINSON, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 358–372, August 1991

004

A NUMERICAL STUDY OF THE LIMITED MEMORY BFGS METHOD AND THE TRUNCATED-NEWTON METHOD FOR LARGE SCALE OPTIMIZATION*

STEPHEN G. NASH† AND JORGE NOCEDAL‡

Abstract. This paper examines the numerical performances of two methods for large-scale optimization: a limited memory quasi-Newton method (L-BFGS), and a discrete truncated-Newton method (TN). Various ways of classifying test problems are discussed in order to better understand the types of problems that each algorithm solves well. The L-BFGS and TN methods are also compared with the Polak–Ribière conjugate gradient method.

Key words. large scale nonlinear optimization, limited memory method, truncated-Newton method, conjugate-gradient method

AMS(MOS) subject classifications. 65, 49

1. Introduction.

Suppose we had to solve an unconstrained optimization problem,

$$\min f(x), \tag{1.1}$$

where f is a smooth function, the number of variables n is large, and a subroutine to evaluate $f(x)$ and $\nabla f(x)$ is available. What method should we choose?

The answer will depend on how much knowledge we have about the structure of the objective function, and on the size of the problem. Newton's method using sparse matrix estimation techniques [2], and the partitioned quasi-Newton method of Griewank and Toint [8] can be highly efficient if sufficient information is supplied to the algorithms. However, it is sometimes too difficult for the user to provide this information, and efficient and reliable software that supplies it automatically is not yet available. In addition, the storage and arithmetic costs for these methods can be prohibitive if the Hessian matrix is not very sparse, or if the problem is extremely large. In these cases, it is better to use other, less ambitious, algorithms. The limited memory BFGS method (L-BFGS) and the discrete truncated-Newton method (TN) represent two classes of methods in this category. They use a low and predictable amount of storage, and only require the function and gradient values at each iterate—and no other information about the problem. Both methods have been tested on large problems and their performance appears to be satisfactory.

In this paper we study the relative performance of L-BFGS and TN on a set of 45 large test problems with a number of variables ranging from 100 to 10,000. Our goal is to study the two methods in a controlled environment to highlight the differences between them and to indicate to the reader the types of problems that are well suited to each algorithm. Our study is limited by the collection of test problems we have used; however, we have tried to produce a test set of considerable breadth. Some test problems arise from applications, whereas others are artificial. Some of the

* Received by the editors January 10, 1990; accepted for publication (in revised form) January 8, 1991.

† Operations Research and Applied Statistics Department, George Mason University, Fairfax, Virginia 22030.

‡ Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208. This author was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under contract DE–FG02–87ER25047–A001, and by National Science Foundation grant ASC 87–19583.

problems are quadratic or nearly quadratic, while others are highly nonlinear. Some are nonconvex. There are varying degrees of ill-conditioning. Our conclusions can only be based on this sample of results, but we hope that they may be more generally useful—both to readers who solve practical problems, and to readers who develop and test new algorithms.

Many limited memory methods have been proposed; some resemble the conjugate-gradient method, and others, the BFGS method. We chose the L-BFGS method because the studies of Gilbert and Lemaréchal [4] and Liu and Nocedal [10] seem to indicate that it is the best limited memory method available to date. Several implementations of the truncated-Newton method have also been proposed, but few codes are as developed as that of Nash [13], which we have chosen for our tests.

Since the authors of the present paper have been closely involved in the development of the methods tested here, a careful effort was made to run them in their best form. This probably also avoided a conscious or unconscious bias towards one of the methods, while setting the test problems or while interpreting the results. The two codes used for this study are not new; they have both been refined during the last years, and neither was especially modified for these runs. We were aware of the dangers of "tuning" methods to a set of test problems. As a result, the nature of the computational tests was specified in detail in advance: A single, fixed version of each algorithm was provided, with all parameters such as tolerances and stopping conditions chosen beforehand. The set of test problems was also chosen in advance, and a *single* run on each test problem was made. (One exception had to be made. On three runs, the algorithms were unable to satisfy our prespecified convergence tolerance. These three problems were run again with a slightly less stringent convergence tolerance.)

Our results indicate that, for L-BFGS and TN, two properties of the problems can be used to make an informed decision on the choice of method. These are the cost of the function-gradient evaluation and the "degree of nonlinearity" of f (this latter property is defined more precisely in §4).

What follows is a brief outline of the paper. The L-BFGS and TN algorithms are described in §2; the test problems and numerical results are presented in §3. The analysis of the test problems, their classification into types, and the correlation of these types with the two methods, appear in §§4 and 5. In the final part of the paper, we compare the two methods to the Polak–Ribière conjugate-gradient method, to obtain another measure of their efficiency.

2. Description of the TN and L-BFGS methods. The truncated-Newton algorithm and the limited memory BFGS method used in this study have already been published, and hence we will only describe them in outline. The L-BFGS has been incorporated into the Harwell Library under the name VA15, and is described by Liu and Nocedal [10]. The TN method is described by Nash [13] (a more precise description of many parts of the method is in Nash [14]). We now give a brief description of their major components.

2.1. The truncated-Newton algorithm. At each "outer" iteration, an approximate solution is found to the Newton equations

$$G_k p_k = -g_k, \tag{2.1}$$

where $g_k = g(x_k) = \nabla f(x_k)$ is the gradient at the kth iteration, and $G_k = \nabla^2 f(x_k)$ is the Hessian. This is done using an "inner" iteration based on a preconditioned linear

conjugate-gradient method [3]. If indefiniteness in the Hessian is detected, the inner iteration is terminated. The approximate solution p_k (the search direction) is then used in a linesearch to get a new point $x_{k+1} = x_k + \alpha_k p_k$ where $f(x_{k+1}) < f(x_k)$ and $\alpha_k > 0$. More precisely:

(1) The outer iteration is terminated when

$$\|g(x_k)\|_\infty < 10^{-6}(1 + |f(x_k)|). \tag{2.2}$$

(2) The linesearch was performed using the iteration described by Gill and Murray [6]. It is based on cubic interpolation, and is terminated when

$$|g(x_k + \alpha_k p_k)^T p_k| \leq \eta |g(x_k)^T p_k| \tag{2.3}$$

and

$$f(x_k) - f(x_k + \alpha_k p_k) \geq -\mu \alpha_k p_k^T g(x_k), \tag{2.4}$$

with $\eta = 0.25$ and $\mu = 10^{-4}$. An initial guess of $\alpha_k = 1$ is used. We refer to (2.3)–(2.4) as the *strong Wolfe conditions.*

(3) The conjugate-gradient inner iteration is preconditioned by a scaled two-step limited memory BFGS method, with Powell's restarting strategy used to reset the preconditioner periodically. It is based on formula (6.5) in [6]: Let t be the iteration where the last restart occurred. Define $s_1 = x_k - x_t$, $y_1 = g_k - g_t$, $s_2 = x_k - x_{k-1}$, $y_2 = g_k - g_{k-1}$, and let D_k be a scaling matrix. Then the approximation H_k to the Hessian is obtained via

$$U = \Gamma_{\text{BFGS}}(D_k, y_1, s_1), \qquad H_k = \Gamma_{\text{BFGS}}(U, y_2, s_2), \tag{2.5}$$

where $\Gamma_{\text{BFGS}}(A, y, s)$ represents the formula for the BFGS update initialized with matrix A, and with vectors y and s corresponding to the change in the gradient and parameter vectors, respectively. The scaling matrix D_k used here is a diagonal approximation to the Hessian obtained by BFGS updating; see equation (10) in Nash [14]. This reference also describes the preconditioner in more detail. We note, in particular, that safeguarding is needed to ensure that the scaling matrix is sufficiently positive definite.

(4) The matrix-vector products required by the inner conjugate-gradient algorithm are obtained by finite differencing [19]. Given a vector v, the product $G_k v$ is approximated by

$$G_k v \approx [g(x_k + \delta v) - g_k]/\delta, \tag{2.6}$$

where $\delta = (1 + \|x_k\|_2)\sqrt{\epsilon}$, and ϵ is the relative machine precision (the "machine epsilon").

(5) The inner algorithm is normally terminated when

$$i\left(1 - \frac{Q_k(p_{i-1})}{Q_k(p_i)}\right) \leq 0.5, \tag{2.7}$$

where i is the counter for the inner iteration, p_i is the ith approximation to the search direction, and $Q_k(p) = \frac{1}{2} p^T G_k p + p^T g_k$ is the value of the quadratic model. This test is explained further in [16]; it guarantees a linear rate of convergence for the optimization algorithm, and detects convergence of the

inner iteration. An upper limit of 50 inner iterations is imposed (this limit was encountered on five of the problems). In addition, the inner algorithm is terminated whenever nonpositive-definiteness is detected in the Hessian. In this case, p_k is the iterate obtained prior to detecting indefiniteness.

The truncated-Newton method requires storage for 16 vectors of length n (the vectors x_k, $g(x_k)$, and 14 work vectors). Each outer iteration consists of three stages: setup, computation of the search direction, and the linesearch. These costs are listed in Table 1. The typical cost was obtained by assuming 1 iteration in the linesearch and 5 inner iterations; these are typical values observed in our test runs.

TABLE 1
Cost of truncated-Newton iteration.

Operation	Flops	f-g	Storage
Setup	$81n$	0	
Search direction iteration	$48n$	1	$16n$
Line search iteration	$4n$	1	
Typical cost	$325n$	6	

Here "Flops" denotes additions, multiplications, or divisions, and "f-g" refers to the number of function-gradient evaluations. The setup is performed once per outer iteration. The search direction and linesearch iterations are repeated until appropriate convergence tests are satisfied, although the linesearch is frequently terminated after only one iteration. Note that each inner iteration requires one gradient evaluation to estimate the matrix-vector product (2.6). The number of inner iterations can vary greatly from problem to problem; thus large deviations from this "typical cost" occur in practice. On the test problems in this paper, where an upper limit of 50 inner iterations was imposed, the number of inner iterations varied between 1 and 50 with both extremes achieved.

2.2. The limited memory BFGS method. We view the L-BFGS method as an adaptation of the BFGS method to large problems, and the implementation of both methods is very similar. In the BFGS method, the approximation H_k to the inverse Hessian matrix of f is updated by

$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T, \tag{2.8}$$

where

$$V_k = I - \rho_k y_k s_k^T, \tag{2.9}$$

$s_k = x_{k+1} - x_k$, $y_k = g_{k+1} - g_k$, and $\rho_k = 1/y_k^T s_k$. The search direction is given by

$$p_{k+1} = -H_{k+1} g_{k+1}. \tag{2.10}$$

We say that H_{k+1} is obtained by applying one BFGS correction to H_k. In the L-BFGS method, instead of forming the matrices H_k, we save the vectors s_k and y_k that define them implicitly. We choose a number m of corrections that we wish to store, and at the kth iteration (with $k > m$) proceed as follows:

(1) Define the diagonal matrix

$$H_k^0 = \frac{y_k^T s_k}{\|y_k\|_2^2} I, \tag{2.11}$$

where I denotes the identity matrix. This scaling is suggested by Oren and Spedicato [20].

(2) Obtain H_k by applying m BFGS corrections to H_k^0, using the m previous vectors s_i and y_i. Compute the product $H_k g_k$ using the recursive formula described in §4 of [18]. This formula takes advantage of the symmetry of the BFGS updating to reduce the number of arithmetic operations.
(3) Perform a linesearch along the direction p_k, enforcing the strong Wolfe conditions (2.3)–(2.4), and trying the steplength $\alpha_k = 1$ first. We use the values $\eta = 0.9$ and $\mu = 10^{-4}$. The linesearch is performed by means of the routine CVSRCH of Moré and Thuente [11], which uses cubic interpolation.
(4) The iteration is finished when (2.2) is satisfied.

We note that once m is chosen, the only parameters in the L-BFGS method are the constants η and μ. The simple scaling of step 1 contributes significantly to the efficiency of the method. Our experience indicates that values of m in the range $3 \leq m \leq 7$ give the best results, and in this paper we use the value $m = 5$. The L-BFGS method requires $2m(n+1) + 4n$ storage locations. Table 2 gives the storage and computational requirements of the L-BFGS method, assuming $m = 5$. We have observed that L-BFGS requires an average of 1.2 iterations within the linesearch; using this value we obtained the typical cost of the iteration, given in Table 2.

TABLE 2
Cost of the limited memory BFGS *iteration.*

Operation	Flops	f-g	Storage
Setup	$4n$	0	
Search direction computation	$33n$	0	$14n$
Linesearch iteration	$4n$	1	
Typical cost	$42n$	1.2	

Except for the first $m - 1$ iterations, the cost of computing the search direction in the L-BFGS method is uniform and predictable; it is a function of m and n.

Comparing Tables 1 and 2, we see that the storage requirements of TN and L-BFGS (with $m = 5$) are very similar, and that the arithmetic costs are drastically different. TN uses an elaborate, variable-cost iteration with partial second-derivative information, whereas L-BFGS uses a fixed, low-cost formula requiring no extra derivative information. In fact, TN *contains* a limited memory method, since the preconditioner used for the inner iteration is similar to an L-BFGS matrix with $m = 2$, and several multiplications with this matrix are performed in one iteration. TN and L-BFGS therefore use different principles to compute search direction. It should be noted that the routines used for the linesearch are similar: both use cubic interpolation to obtain the strong Wolfe conditions (only the value of the parameter η in (2.3) is different).

3. Numerical tests. All the problems used in our tests have been described elsewhere; approximately half of them were used by Liu and Nocedal in their study of L-BFGS [10]. Table 3 lists the problems and the number of variables used for the runs, and gives references to detailed descriptions of the test functions and starting points. For test problems 8, 9, and 10, starting point 3 from the reference was used. The problems are not numbered consecutively because they belong to a larger collection of test problems to which we may want to refer in future studies. The number of variables in the test set ranges from 100 to 10,000, and, as will be seen in §4, the problems form a varied grouping. We verified that, in each run, both methods converged to the same solution point.

The results of the tests are given in Tables 4 and 5. There "It" and "f-g" record

TABLE 3
List of test functions

Problem	Name	Reference	n
1	Calculus of variations 1	Gill and Murray [5]	100, 200
2	Calculus of variations 2	Gill and Murray [5]	100, 200
3	Calculus of variations 3	Gill and Murray [5]	100, 200
6	Generalized Rosenbrock	Moré, Garbow, and Hillstrom [12]	100, 500
8	Penalty 1	Gill and Murray [6]	100, 1000
9	Penalty 2	Gill and Murray [6]	100
10	Penalty 3	Gill and Murray [6]	100, 1000
28	Extended Powell singular	Moré, Garbow, and Hillstrom [12]	100, 1000
29	Variably dimensioned	Moré, Garbow, and Hillstrom [12]	100, 500
31	Brown almost linear	Moré, Garbow, and Hillstrom [12]	100, 200
38	Tridiagonal 1	Buckley and LeNir [1]	100, 1000
39	Linear minimal surface	Toint [23]	121, 961
40	Boundary-value problem	Toint [23]	100
41	Broyden tridiagonal nonlinear	Toint [23]	100
42	Extended ENGVL1	Toint [23]	1000, 10,000
43	Ext. Freudenstein and Roth	Toint [23]	100, 1000
45	Wrong extended Wood	Toint [23]	100
46(1)	Matrix square root ($ns = 1$)	Liu and Nocedal [9]	100
46(2)	Matrix square root ($ns = 2$)	Liu and Nocedal [9]	100
47	Sparse matrix square root	Liu and Nocedal [9]	100, 1000
48	Extended Rosenbrock	Moré, Garbow, and Hillstrom [12]	1000, 10,000
49	Extended Powell	Moré, Garbow, and Hillstrom [12]	100, 1000
50	Tridiagonal 2	Toint [23]	100, 1000
51	Trigonometric	Moré, Garbow, and Hillstrom [12]	100, 1000
52	Penalty 1 (2nd version)	Moré, Garbow, and Hillstrom [12]	1000, 10,000
53	INRIA u1ts0	Gilbert and Lemaréchal [4]	403

the iteration count and the number of function-gradient evaluations, respectively. For the truncated-Newton method, this includes the number of gradient evaluations of the inner iteration (one gradient computation is considered to be as expensive as a simultaneous function and gradient evaluation). The times are measured in seconds and reflect total execution time, including function-gradient evaluations. The runs were made on one processor of an Encore Multimax computer, using Fortran in double precision (about 16 decimal digits). An upper bound of 9,999 function-gradient evaluations was imposed, but was only encountered by L-BFGS on test function one; this is indicated by a "+" next to the function-gradient count. Most of the test functions were run for two values of n. Table 4 presents the results for the smaller dimensions.

Table 5 gives the results for higher dimensions. On three runs, both routines terminated abnormally in the linesearch (a lower point could not be found). This is indicated by a "*." In all three cases the algorithms were close to the solution. These problems were rerun with the convergence tolerance in (2.2) set to 10^{-5}, to make it less stringent. With this change, these runs were successful and are included in Table 5. As before, "+" indicates that the function evaluation limit was reached.

The results of Tables 4 and 5 are summarized in Table 6. "Much better" is defined to be a difference of more than 30 percent in results. Differences of less than 10 percent were considered to be insignificant ("even"), as were absolute differences of less than one second and differences of less than five function-gradient evaluations. A failure is recorded as "much better" performance for the algorithm that succeeded.

What can we conclude from these results? First, neither algorithm is clearly superior to the other. Second, the limited-memory BFGS method tends to use fewer

TABLE 4

Smaller test problems. Comparison of the limited memory BFGS *method* ($m = 5$) *with the truncated-Newton method.*

		L-BFGS			TN		
P	N	It	f-g	Time	It	f-g	Time
1	100	9707	+9999	2380.0	28	466	105.0
2	100	1605	1669	304.0	27	242	41.4
3	100	3095	3216	640.0	45	325	60.8
6	100	257	291	19.0	78	683	35.3
8	100	30	36	2.1	4	26	0.8
9	100	21	23	2.4	8	48	4.4
10	100	82	90	7.1	20	107	7.3
28	100	57	67	3.6	14	70	3.0
29	100	36	37	2.4	9	42	1.3
31	100	24	27	10.7	14	101	38.5
38	100	120	128	8.5	18	77	4.4
39	121	66	70	8.4	18	187	19.6
40	100	2219	2296	188.0	55	2091	146.0
41	100	28	31	2.1	14	77	4.7
42	1000	15	17	10.2	15	75	41.0
43	100	19	21	1.7	10	38	2.9
45	100	48	56	3.8	14	59	3.3
46(1)	100	362	377	75.9	30	339	61.9
46(2)	100	438	453	92.8	35	556	101.0
47	100	87	95	10.1	17	94	8.8
48	1000	38	49	26.4	16	79	32.0
49	100	57	67	3.8	14	70	3.0
50	100	129	133	9.5	17	78	4.2
51	100	53	60	8.9	28	233	30.2
52	1000	5	6	2.3	3	10	4.3

TABLE 5

Larger test problems. Comparison of the limited memory BFGS *method* ($m = 5$) *with the truncated-Newton method.*

		L-BFGS			TN		
P	N	It	f-g	Time	It	f-g	Time
1	200	9695	+9999	5040.0	38	929	428.0
2	200	1734	1785	646.0	37	456	154.0
3	200	7248	7482	2960.0	76	599	220.0
6	500	1054	1177	389.0	356	3446	796.0
8	1000	30	34	21.0	12	58	23.9
10	1000	103	*114	89.4	30	*200	142.0
28	1000	54	61	35.8	15	75	34.6
29	500	48	*49	15.4	12	*54	8.8
31	200	3	4	5.8	4	20	24.9
38	1000	405	423	302.0	33	208	121.0
39	961	165	172	187.0	27	387	364.0
42	10,000	14	17	102.0	24	111	696.0
43	1000	16	20	15.4	15	75	59.1
47	1000	145	157	168.0	22	160	145.0
48	10,000	37	50	261.0	29	118	639.0
49	1000	54	61	35.6	14	68	31.8
50	1000	457	476	328.0	30	210	118.0
51	1000	46	*57	80.1	35	*370	467.0
52	10,000	6	8	32.5	5	19	90.9
53	403	57	63	410.0	14	100	656.0

TABLE 6
Summary of results: number of problems for which a method was better in terms of function-gradient evaluations and time.

Result	f-g	Time
L-BFGS much better	18	21
L-BFGS better	6	1
even	8	4
TN better	2	1
TN much better	11	18

function-gradient evaluations. Third, in terms of time, neither algorithm is a clear winner: the higher iteration cost of TN is compensated by a much lower iteration count, on the average. Fourth, the truncated-Newton method solved all the problems, whereas the limited memory BFGS method failed twice (on both versions of the first test problem). In §5 we analyze these results further, and present more detailed conclusions.

Tests of a simulated Newton method were also made. Algorithm TN was modified by changing the termination rule for the inner algorithm from (2.7) to

$$\frac{\|Gp_i + g_k\|_2}{\|g_k\|_2} \leq 10^{-8},$$

and with an upper limit of $\min(n, 500)$ inner iterations imposed. Even though this does not represent a well-designed Newton method for large scale problems, it can indicate how many outer iterations would be required if the costs of the inner algorithm could be ignored. We refer to this simulated Newton method as SN. On the smaller test problems, SN required 313 iterations and TN required 551; in the worst case TN required five times more iterations. On the larger test problems, SN required 598 iterations and TN required 828; in the worst case TN required seven times more iterations. Hence, over the entire test set, TN required about 50 percent more iterations than SN. The L-BFGS method, whose iteration is much cheaper, typically requires between 3 and 10 times more iterations than TN.

We conclude this section with a few comments on the partitioned quasi-Newton method of Griewank and Toint [8]. It assumes that the objective function is partially separable, i.e., that it can be written as

$$f(x) = \sum_{i=1}^{ne} f_i(x),$$

where the ne element functions f_i depend only on a few variables. The partitioned quasi-Newton method (PQN) takes advantage of this structure of f by updating a quasi-Newton approximation to each of the element functions f_i. Liu and Nocedal [10] compared L-BFGS and PQN, and concluded that when the number of variables entering into the element functions is very small (say, less than four), PQN is vastly superior to L-BFGS, both in terms of function evaluations and time. However, if the element functions f_i depend on four or five variables, L-BFGS and PQN are often comparable. Furthermore, if more than five or six variables enter into the element functions, L-BFGS is likely to be more efficient.

4. Classification of the test problems. We would like to better understand the types of problems that each algorithm solves well. This immediately raises the

question of how to classify the test problems. Among the various function characteristics that are relevant to the convergence theory or computational behavior of algorithms, we have selected the following.

(1) **Deviation from quadratic.** To assess this, we use the Taylor series approximation of the gradient, and define

$$\mathrm{DQ} = \frac{\|g(x_0) - g(x^*) - G(x^*)p\|_\infty}{\|p\|_\infty^2}, \tag{4.1}$$

where x_0 and x^* are the starting point and the solution, and $p = x_0 - x^*$. Since we have scaled the difference by $\|p\|_\infty^2$, DQ gives a measure of the size of the third derivatives.

(2) **Condition number of the Hessian.** Using the l_2-norm, we measure the condition numbers of the Hessian at the starting point and solution point; these are denoted by K_0 and K_*, respectively.

(3) **Convexity.** This was sometimes determined by observation; otherwise we performed several computations to guide us. We computed the eigenvalues of the Hessian at x_0. If any were negative, the problem was clearly nonconvex. In addition, the inner algorithm of the truncated-Newton method can detect indefiniteness (although it is not guaranteed to find it). If none of these indicators suggested indefiniteness, and if we were not able to ascertain this theoretically, the problem was labeled "presumably convex."

(4) **Eigenvalue structure.** We plotted the eigenvalue distribution of the Hessian at the starting point and final point. The graphs are not given here due to space limitations, but can be obtained from the authors. The eigenvalue distribution greatly affects the performance of the inner conjugate-gradient algorithm of the truncated-Newton method, and Gill and Murray [6] suggest that it is also related to the efficiency of limited memory methods.

The characteristics of most of our test problems are displayed in Table 7. We have only considered the case when $n \approx 100$, because the eigenvalue analysis for values of n in the thousands is not tractable. In particular, test problem 42 was analyzed for $n = 100$ rather than $n = 1000$ (TN solved this smaller problem in 1.8 seconds using nine iterations and 35 function-gradient evaluations; L-BFGS used .9 seconds, 14 iterations, and 15 function-gradient evaluations). In Table 7, we also indicate which of the two methods performed best, and give the percentage difference in performance.

The next table has been arranged to show that there is a strong correlation between the degree of nonlinearity of the function and the success of the methods. We have concentrated on those problems of Table 7 in which the performance of the two methods is markedly different. Table 8 lists the problems for which one of the methods was better by at least 30 percent in terms of *both* time and function-gradient evaluations. The entries are ordered by degree of nonlinearity DQ.

We now list, in Table 9, three other properties of the test functions that help measure problem complexity, even though they do not appear to be directly useful for purposes of analysis.

(1) **Sparsity.** The degree of sparsity and the sparsity pattern determine whether Newton's method with sparse matrix techniques is attractive. Neither TN nor L-BFGS takes advantage of sparsity.

(2) **Cost of evaluating f and g.** Here we merely timed the function values for most problems by averaging the time for 1000 function-gradient evaluations. For problem 53, only 20 evaluations were used in the timing because of the expense of computing the function.

TABLE 7
Problem characteristics.

P	N	DQ	$1/K_0$	$1/K_*$	Convexity	Winner	Percent difference f-g	Percent difference Time
1	100	10^{-2}	10^{-8}	10^{-8}	presumed	TN	2045	2167
2	100	10^{0}	10^{-5}	10^{-5}	presumed	TN	590	634
3	100	10^{0}	10^{-6}	10^{-6}	presumed	TN	890	953
6	100	10^{2}	10^{-3}	10^{-4}	no	L-BFGS	135	86
8	100	10^{2}	10^{-1}	10^{-2}	no	TN	38	163
9	100	10^{5}	10^{-3}	10^{-2}	presumed	L-BFGS	109	83
10	100	10^{2}	10^{-2}	10^{-3}	presumed	L-BFGS	18	3
28	100	10^{1}	10^{-3}	10^{-8}	presumed	even	4	20
29	100	10^{13}	10^{-9}	10^{-6}	no	even	14	85
31	100	10^{2}	0	10^{-8}	presumed	L-BFGS	274	260
38	100	10^{-2}	0	0	yes	TN	66	93
39	121	10^{1}	10^{-9}	0	no	L-BFGS	167	133
40	100	10^{-2}	10^{-7}	10^{-7}	no	TN	10	30
41	100	10^{1}	0	0	no	L-BFGS	148	124
42	100	10^{1}	10^{-1}	10^{-1}	presumed	L-BFGS	133	100
43	100	10^{0}	10^{-3}	10^{-4}	presumed	L-BFGS	81	71
45	100	10^{3}	10^{-3}	10^{-2}	no	even	5	15
46(1)	100	10^{1}	10^{-4}	10^{-4}	no	TN	11	23
46(2)	100	10^{1}	10^{-3}	10^{-4}	no	L-BFGS	23	9
47	100	10^{1}	10^{-2}	10^{-2}	no	even	1	15
49	100	10^{1}	10^{-3}	10^{-7}	presumed	even	4	27
50	100	10^{-4}	0	0	yes	TN	58	126
51	100	10^{2}	10^{-3}	10^{-2}	no	L-BFGS	288	239
53	403	10^{0}	0	0	no	L-BFGS	59	60

TABLE 8
Selected test problems, ordered by degree of nonlinearity.

P	N	DQ	Winner	Percent difference f-g	Percent difference Time
9	100	10^{5}	L-BFGS	109	83
31	100	10^{2}	L-BFGS	274	260
8	100	10^{2}	TN	38	163
6	100	10^{2}	L-BFGS	135	86
51	100	10^{2}	L-BFGS	288	239
41	100	10^{1}	L-BFGS	148	124
42	100	10^{1}	L-BFGS	133	100
39	121	10^{1}	L-BFGS	167	133
3	100	10^{0}	TN	890	953
2	100	10^{0}	TN	590	634
43	100	10^{0}	L-BFGS	81	71
53	403	10^{0}	L-BFGS	59	60
1	100	10^{-2}	TN	2045	2167
38	100	10^{-2}	TN	66	93
50	100	10^{-4}	TN	58	126

(3) Laboriousness. This is an attempt to measure the difficulty of solving a problem. We considered both algorithms, and recorded the minimum number of function-gradient evaluations and the minimum time to solve the problem.

One final point about the test problems should be made. Of those analyzed here, 13 out of 24 (and all eight of the most laborious problems) are singular, nearly singular, or nonconvex. The fact that all these problems were successfully solved by both methods seems to indicate that their implementations are very robust.

TABLE 9
Other problem characteristics.

P	N	Sparsity	Cost	Laboriousness		Winner	Percent difference	
				f-g	Time		f-g	Time
1	100	7-diagonal	.155	466	105.0	TN	2045	2167
2	100	7-diagonal	.123	242	41.4	TN	590	634
3	100	7-diagonal	.121	325	60.8	TN	890	953
6	100	tridiagonal	.014	291	19.0	L-BFGS	135	86
8	100	dense	.011	26	0.8	TN	38	163
9	100	dense	.054	23	2.4	L-BFGS	109	83
10	100	dense	.026	90	7.1	L-BFGS	18	3
28	100	4 × 4 block diagonal	.007	67	3.0	even	4	20
29	100	dense	.009	37	1.3	even	14	85
31	100	dense	.322	27	10.7	L-BFGS	274	260
38	100	tridiagonal	.014	77	4.4	TN	66	93
39	121	9 non-zero diagonals	.054	70	8.4	L-BFGS	167	133
40	100	5-diagonal	.026	2091	146.0	TN	10	30
41	100	5-diagonal	.018	31	2.1	L-BFGS	148	124
42	100	tridiagonal	.013	15	0.9	L-BFGS	133	100
43	100	tridiagonal	.032	21	1.7	L-BFGS	81	71
45	100	tridiagonal	.015	56	3.3	even	5	15
46(1)	100	dense	.137	339	61.9	TN	11	23
46(2)	100	dense	.131	453	92.8	L-BFGS	23	9
47	100	9-diagonal	.046	94	8.8	even	1	15
49	100	4 × 4 block diagonal	.009	67	3.0	even	4	27
50	100	tridiagonal	.011	78	4.2	TN	58	126
51	100	dense	.094	60	8.9	L-BFGS	288	239
53	403	dense	5.980	63	410.0	L-BFGS	59	60

5. Further analysis of the numerical results. Using the tables of results and of function characteristics of the previous sections, we now try to correlate the types of test problems with the success of TN and L-BFGS. Since we have seen that in terms of time the two methods are similar, we concentrate on the number of function-gradient evaluations.

The first, very visible, trend is that the performance of the two algorithms appears to be correlated with the degree of nonlinearity DN: for quadratic and approximately quadratic problems, TN outperforms L-BFGS. In fact, TN was better almost exclusively for these types of problems. Among the 11 problems for which TN was considered "much better" in Table 6, four are quadratic (both versions of problems 38 and 50) and six are approximately quadratic (both versions of problems 1, 2, and 3). Moreover, one of the two problems for which TN was considered "better" in this table is approximately quadratic (problem 40). For most of the highly nonlinear problems, L-BFGS performed better. It appears that TN's effort to approximate the Newton step is not paying off on highly nonlinear problems. Continuing work on trying to improve the performance of TN by reducing the number of inner iterations on problems that are detected to be highly nonlinear will be reported in a future paper [17].

A study of the eigenvalue distribution of the Hessian matrices shows no clear correlation between the success of the methods and the eigenvalue structure. For problems with ill-conditioned Hessians, TN is perhaps better, but this is difficult to ascertain from our tests. Clustering of eigenvalues at the solution does not seem to benefit one method more than the other. There is little or no correlation between the other measures of the test problems and the performance of the two methods.

L-BFGS did very poorly on the calculus of variations problems 1, 2, and 3. These

problems are not only ill conditioned; their Hessians have many eigenvalues near zero (for problem 1 for $n = 100$, about half the eigenvalues are near zero). We ran L-BFGS with a more accurate linesearch and using more memory, and even though the performance improved substantially, it was still far from competitive with TN. For example, using $m = 20$ corrections and setting $\eta = 10^{-3}$ in (2.3) resulted in 418 iterations, 861 function-gradient evaluations, and 183 seconds for problem 2 with $n = 100$. The calculus of variations problems also caused difficulties for earlier versions of the TN method that used no preconditioning; the performance was as bad as for L-BFGS. If n is increased to 1000, TN has trouble solving the problem; however, block versions of the TN method (requiring greater storage and work per iteration) are capable of solving it [15]. We should note that Newton's method is very effective on this problem, suggesting that an accurate approximation to the Newton direction is needed.

It appears from this test set that, in terms of function evaluations, L-BFGS is preferable to TN for more highly nonlinear (not approximately quadratic) problems. However, TN almost always requires many fewer iterations than L-BFGS, and therefore, if the number of gradient evaluations in the inner iteration could be significantly reduced, TN would be competitive or more efficient than L-BFGS. How to realize these savings is a question that deserves further investigation. It does not appear that automatic differentiation (see, for example, [7]) will help, because the cost is of the same order as that of the gradient differences used here. Differencing along selective directions determined by the sparsity of the problem could be useful, but in this case both Newton's method with sparse matrix techniques and the partitioned quasi-Newton method may be preferable.

6. The nonlinear conjugate-gradient method. Now that we have studied the relative performances of L-BFGS and TN, we will use another, well-established, algorithm to measure their efficiency. To this end, we will solve our set of test problems with the Polak–Ribière version of the conjugate-gradient method, which is one of the classical methods for solving large problems. We chose the Polak–Ribière method for the following reason.

More recent implementations of the conjugate-gradient method, like the Harwell routine VA14 of Powell [21] or the routine CONMIN of Shanno and Phua [22], which include automatic restarts, and store additional information, require fewer function evaluations than the Polak–Ribière method. However, Liu and Nocedal [10] found that L-BFGS clearly outperforms CONMIN, both in terms of computer time and function evaluations—and it is known that CONMIN is more efficient than VA14. On the other hand, the Polak–Ribière method appeared to be often competitive with L-BFGS in terms of computer time (but not in terms of function evaluations). Therefore, we chose the Polak–Ribière method for this study, since it appears to be the only implementation of the conjugate-gradient method that could be competitive (at least in terms of computer time) with TN and L-BFGS.

The Polak–Ribière iteration is

$$x_{k+1} = x_k + \alpha_k p_k, \tag{6.1}$$

where the steplength α_k satisfies the strong Wolfe conditions (2.3)–(2.4), and where

$$p_k = -g_k + \beta_k p_{k-1}, \tag{6.2}$$

TABLE 10
Smaller problems. Results of the Polak–Ribière conjugate-gradient method (CG).

		CG			L-BFGS		TN	
P	N	It	f-g	Time	f-g	Time	f-g	Time
1	100	4915	+9999	1910.0	≈	≈		
2	100	656	1506	201.0	+	*		
3	100	2082	4243	641.0		≈		
6	100	275	1129	24.2				*
8	100	9	28	0.6	+	*		≈
9	100	F						
10	100	119	304	11.3				
28	100	108	277	4.7				
29	100	7	51	1.0		*		≈
31	100	3	12	4.3	+	*	+	*
38	100	76	154	3.8		*		≈
39	121	63	180	12.6			≈	*
40	100	5001	+9999	363.0				
41	100	27	63	1.8		≈	+	*
42	1000	14	37	8.0		*	+	*
43	100	54	141	7.1				
45	100	36	120	2.8		*		≈
46(1)	100	286	583	89.3				
46(2)	100	313	639	99.4		≈		≈
47	100	73	152	9.6		≈		≈
48	1000	25	108	14.2		*		*
49	100	104	255	4.8				
50	100	79	160	3.6		*		≈
51	100	54	114	12.1			+	*
52	1000	4	10	1.6	≈	≈	≈	*

with

$$\beta_k = \frac{y_{k-1}^T g_k}{\|g_{k-1}\|^2}. \tag{6.3}$$

The linesearch is performed by means of the routine of Moré and Thuente [11], previously mentioned in §2, with two small changes: (i) Since the Wolfe conditions do not ensure that descent directions are always generated, we continue the linesearch iteration until the descent condition is guaranteed. (ii) We insist that the linesearch performs at least one quadratic (or cubic) interpolation, and hence the algorithm reduces to the linear conjugate-gradient algorithm if the objective function is quadratic.

We chose $\mu = 10^{-4}$ and $\eta = 0.1$, for the parameters in (2.3)–(2.4), because our earlier experience with the Polak–Ribière method indicated that these values give the best results. The stopping condition was (2.2). The tests were performed, as before, on an Encore Multimax. The algorithm was restarted every n iterations by setting $\beta_k = 0$, which ensures global convergence in exact arithmetic. We note that n is often larger than the number of iterations, so that the algorithm performs no restarts in many of our test runs. The Polak–Ribière method requires $4n$ storage locations, and as shown by (6.2)–(6.3), the computation of d_k is very inexpensive.

Tables 10 and 11 present the results of the Polak–Ribière (CG) method on the whole collection of test problems. Under the column labelled "CG" we give the number of iterations/number of function-gradient evaluations, and the total computing time. As before, a "+" indicates that the function evaluation limit was reached, and a "*" indicates that the weaker convergence test (described in the third paragraph of §3) was used. In the runs marked "F" the search direction was so out of scale that

TABLE 11
Larger problems. Results of the Polak–Ribière conjugate-gradient method (CG).

		CG			L-BFGS		TN	
P	N	It	f-g	Time	f-g	Time	f-g	Time
1	200	4804	+9999	3840.0	≈	≈		
2	200	1106	2491	657.0		≈		
3	200	3336	6847	2060.0	≈	*		
6	500	1098	4889	515.0				*
8	1000	13	36	7.3	≈	*	+	*
10	1000		*F					
28	1000	167	456	72.1				
29	500	8	*98	7.1		*		*
31	200	3	14	20.3			+	*
38	1000	285	573	138.0		*		
39	961	161	519	319.0				*
42	10000	11	33	68.9		*	+	*
43	1000	16	56	23.3			+	*
47	1000	145	302	181.0		≈		
48	10000	18	76	103.0		*	+	*
49	1000	225	615	104.0				
50	1000	294	591	116.0		*		≈
51	1000	41	*91	93.7			+	*
52	10000	5	13	21.0	≈	*	+	*
53	403	62	144	926.0				

overflow occurred. Additional safeguarding could remedy these failures, but was not implemented. We compare the performance of CG with that of L-BFGS and TN by indicating the problems for which CG was more effective. In the column labelled "L-BFGS," a "+" in the column "f-g" indicates that CG required fewer function calls, and a "*" in the column "Time" indicates that it required less computing time than L-BFGS. The sign "≈" means equal or near-equal performance (measured in the same way as in Table 6: differences of less than 10 percent, less than one second, or less than five gradient evaluations were considered negligible). The same notation is used in the column labelled "TN." We verified, for each run, that the solution point obtained by CG coincided with that found by L-BFGS and TN.

It is clear that the conjugate-gradient method is not competitive with TN or L-BFGS in terms of function evaluations. However, it is often efficient in terms of computing time, due to its very low iteration cost. From this set of test runs, we conclude that the conjugate-gradient method is not to be recommended when the function evaluation is expensive, but that it may be useful for very large problems whose objective functions are relatively inexpensive.

Acknowledgments. We would like to thank the referees for several valuable comments and suggestions.

REFERENCES

[1] A. BUCKLEY AND A. LENIR (1983), QN-*like variable storage conjugate gradients*, Math. Programming, 27, pp. 155–175.

[2] T.F. COLEMAN AND J. MORÉ (1984), *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, 28, pp. 243–270.

[3] P. CONCUS, G. GOLUB, AND D. P. O'LEARY (1976), *A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations*, in J. Bunch and D. Rose, eds., in Sparse Matrix Computations, Academic Press, New York, pp. 309–332.

[4] J. C. GILBERT AND C. LEMARÉCHAL (1989), *Some numerical experiments with variable storage quasi-Newton algorithms*, Math. Programming, 45, pp. 407–435.

[5] P. E. GILL AND W. MURRAY (1973), *The numerical solution of a problem in the calculus of variations*, in Recent Mathematical Developments in Control, D.J. Bell, ed., Academic Press, New York, pp. 97–122.

[6] ———(1979), *Conjugate-gradient methods for large-scale nonlinear optimization*, Report SOL 79–15, Department of Operations Research, Stanford University, Stanford, CA.

[7] A. GRIEWANK (1989), *On automatic differentiation*, in Mathematical Programming, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers, Tokyo, pp. 83–107.

[8] A. GRIEWANK AND PH. L. TOINT (1982), *Partitioned variable metric updates for large structured optimization problems*, Numer. Math., 39, pp. 119–137.

[9] D. C. LIU AND J. NOCEDAL (1988), *Test results of two limited memory methods for large scale optimization*, Report NAM 04, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois.

[10] ———(1989), *On the limited memory* BFGS *method for large scale optimization*, Math. Programming, 45, pp. 503–528.

[11] J. J. MORÉ, AND D. J. THUENTE (1990), *On linesearch algorithms with guaranteed sufficient decrease*, Mathematics and Computer Science Division Preprint MCS-P153-0590, Argonne National Laboratory, Argonne, IL.

[12] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM (1981), *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7, pp. 17–41.

[13] S. G. NASH (1984), *User's guide for* TN/TNBC: *Fortran routines for nonlinear optimization*, Report 397, Mathematical Sciences Department, The Johns Hopkins University, Baltimore, MD.

[14] ———(1985), *Preconditioning of truncated-Newton methods*, SIAM J. Sci. Statist. Comput., 6, pp. 599–616.

[15] S. G. NASH AND A. SOFER (1989), *Block truncated-Newton methods for large-scale nonlinear optimization*, Math. Programming, 45, pp. 529–546.

[16] ———(1990a), *Assessing a search direction within a truncated-Newton method*, Oper. Research Lett., 9, pp. 219–221.

[17] ———(1990b), *A practical truncated-Newton method for parallel optimization*, Report 63, Center for Computational Statistics, George Mason University, Fairfax, VA.

[18] J. NOCEDAL (1980), *Updating quasi-Newton matrices with limited storage*, Math. Comp., 35, pp. 773–782.

[19] D. P. O'LEARY (1983), *A discrete Newton algorithm for minimizing a function of many variables*, Math. Programming, 23, pp. 20–33.

[20] S. OREN AND E. SPEDICATO (1976), *Optimal conditioning of self-scaling variable metric algorithms*, Math. Programming, 10, pp. 70–90.

[21] M. J. D. POWELL (1977), *Restart procedures for the conjugate gradient method*, Math. Programming, 12, pp. 241–254.

[22] D. F. SHANNO AND K. H. PHUA (1980), *Remark on algorithm* 500: *Minimization of unconstrained multivariate functions*, ACM Trans. Math. Software, 6, pp. 618–622.

[23] PH.L. TOINT (1983), *Test problems for partially separable optimization and results for the routine* PSPMIN, Report Nr 83/4, Department of Mathematics, Facultés Universitaires de Namur, Namur, Belgium.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 373–400, August 1991

005

MASSIVELY PARALLEL ROW-ACTION ALGORITHMS FOR SOME NONLINEAR TRANSPORTATION PROBLEMS*

STAVROS A. ZENIOS† AND YAIR CENSOR‡

Abstract. Row-action iterative algorithms are developed for two classes of nonlinear optimization problems with transportation constraints: entropy problems where the objective function is a sum of $x[\ln(x/a) - 1]$ terms, and quadratic problems where the objective function is a sum of $\frac{1}{2}wx^2 + cx$ terms. The algorithms are specialized to take advantage of both the structure of the transportation constraints and the special forms of the objective functions. Both generalized and pure networks are dealt with in this paper.

The algorithms are well suited for parallel computing. Implementations are developed on a massively parallel Connection Machine CM-2 with up to 32K processing elements. The algorithms solve test problems with 4000 nodes and 4 million arcs in less than one minute of computer time. On a maximally configured machine with 64K processing elements, they achieve a peak computing rate of 3 GFLOPS.

Key words. transportation models, nonlinear programming, row-action algorithms, parallel computation

AMS(MOS) subject classifications. primary 90-08, 90B06, 90C06; secondary 65Y05

1. Introduction.

In this paper we consider nonlinear optimization problems with transportation constraints. Such problems appear in several areas of applications, mainly in logistics and transportation planning. Most common instances of matrix-balancing problems are also formulated over transportation constraints. A significant body of literature exists on the solution of linear transportation problems that dates back to Kantorovich [15]. For an early account, see Ford and Fulkerson [12]. For recent work, see Miller, Pekney, and Thompson [19], where a parallel implementation of the transportation simplex algorithm is also developed. A review of models and algorithms is given in Dembo, Mulvey, and Zenios [10].

The development of nonlinear programming algorithms for the same class of problems is a more recent activity. Quite often, it is associated with the development of iterative algorithms for matrix balancing (see, e.g., [2], [13], or [22]). In those instances, some nonlinear function is optimized over the transportation constraints to achieve the balancing of the matrix. Dual algorithms for continuously differentiable strictly convex nonlinear functions over more general transshipment network constraints have been developed by Bertsekas, Hossein, and Tseng [3]. Ohuchi and Kaji [21] developed dual algorithms for quadratic problems over transportation constraints. Newton's algorithm has been specialized for the transportation constraints by Klincewicz [16]. Algorithms for nonlinear generalized networks—i.e., with arc gains—that include as a special case the transportation problem, were published in [1] and [26].

In this paper, we develop specialized algorithms for problems with transportation constraints when the objective function is either the sum of entropy terms of the form

* Received by the editors April 26, 1990; accepted for publication (in revised from) February 25, 1991.

† Decision Sciences Department, Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania 19104. The work of this author was supported in part by National Science Foundation grant CCR-8811135 and Air Force Office of Scientific Research grant 91-0168.

‡ Department of Mathematics and Computer Science, University of Haifa, Haifa 31905, Israel. The work of this author was supported by National Science Foundation grant SES-9100216 and National Institutes of Health grant HL-28438 and was performed while he was visiting the Decision Sciences Department, Wharton School, University of Pennsylvania, and the Medical Imaging Processing Group (MIPG), Department of Radiology, University of Pennsylvania.

$x[\ln(x/a) - 1]$ or quadratic terms of the form $\frac{1}{2}wx^2 + cx$. Both functions belong to the family of Bregman's functions [6] as characterized by Censor and Lent [7]. It is thus possible to develop row-action-type algorithms for those special objective functions, and for linear equality, inequality, or interval constraints. The algorithms can be viewed as coordinate ascent methods for maximizing the dual functional obtained by dualizing both the equality (i.e., flow-conservation) constraints and the bounds on the variables. The iterative procedures we develop here are specializations of the algorithms of Censor and Lent [7] that exploit the special structure of the transportation constraints. For generalized transportation problems with quadratic objective functions, or pure network problems, very simple formulae are obtained for the iterative steps of the algorithms. In the case of generalized networks with entropy objective function, the iterative step of the algorithm would require solution of a nonlinear equation. Even in this case, however, closed-form formulae can be obtained for an approximate solution to this equation that guarantees asymptotic convergence of the algorithm. For this, we draw on results from Censor et al. [8]. The algorithms are well suited for parallel computing, especially on massively parallel systems. This characteristic is indeed our motivation for designing these algorithms in the first place, and implementations are developed on a Connection Machine CM-2 with up to 32K (1K = 1024) processing elements, using simple modifications of the data structures from Zenios [27].

The contributions of this paper are the development of the specialized iterative algorithms for the transportation constraints, and the massively parallel implementations. The algorithms for generalized network problems and the algorithm for pure network problems with quadratic objective appear to be new in this field. Their structure is, however, essentially similar to that of the ART4 algorithm of Herman and Lent [28], which was proposed as an iterative procedure for image reconstruction from projections. ART4 is a development of the earlier Hildrath algorithm [29], [30]. The algorithm for entropy optimization over pure transportation constraints is a generalization of an existing algorithm for matrix balancing. The test problems solved are, to our knowledge, the largest reported in the literature and solution times do not exceed a few seconds of computer time. As subsequent computational results show, the algorithms on the Connection Machine also outperform by a large margin other algorithms for the same problems on more "conventional" parallel or serial computers.

Section 2 formulates the problems and develops the algorithms: we develop the algorithms for generalized network problems and obtain the algorithms for pure networks as a special case. It also provides some general background on the Connection Machine CM-2 and a brief overview of the Paris language. The parallel implementations are discussed in §3, and §4 provides a summary of the computational experiments and an interpretation of the results. Concluding remarks are given in §5.

2. Nonlinear transportation problems.

2.1. Problem formulation. Let $\langle m \rangle$ denote the set $\{1, 2, 3, \cdots, m\}$. Denote by $\Re^m$ the m-dimensional Euclidean space and $\langle \cdot, \cdot \rangle$ is the Euclidean inner product. A transportation graph is defined as the triplet $\mathcal{G} = (V_O, V_D, \mathcal{E})$, where $V_O = \langle m_O \rangle$, $V_D = \langle m_D \rangle$, and $\mathcal{E} \subseteq \{(i,j) \mid i \in V_O, j \in V_D\}$. V_O and V_D are the sets of origin and destination nodes of cardinality m_O and m_D, respectively. $\mathcal{E}$ is the set of n directed arcs (i,j), with origin node i and destination node j, which belong to the graph, $n \leq m_O m_D$. It is not assumed that the graph is dense, i.e., (i,j) may not be in $\mathcal{E}$ for some values of $i \in V_O$ and $j \in V_D$. Some additional notation is needed to define the nonlinear transportation problems. Let

$x = (x_{ij}) \in \Re^n$, $(i,j) \in \mathcal{E}$, be the vector of flows;
$u = (u_{ij}) \in \Re^n$, $(i,j) \in \mathcal{E}$, be the vector of upper bounds on the flows;
$m = (m_{ij}) \in \Re^n_>$, $(i,j) \in \mathcal{E}$, be the vector of positive multipliers on the flows;
$s = (s_i) \in \Re^{m_O}$, $i \in V_O$, be the vector of supplies;
$d = (d_j) \in \Re^{m_D}$, $j \in V_D$, be the vector of demands;
$\pi^O = (\pi^O_i) \in \Re^{m_O}$, $i \in V_O$, be the vector of dual prices for origin nodes;
$\pi^D = (\pi^D_j) \in \Re^{m_D}$, $j \in V_D$, be the vector of dual prices for destination nodes;
$r = (r_{ij}) \in \Re^n$, $(i,j) \in \mathcal{E}$, be the vector of dual prices for the bound constraints;
$\delta^+_i = \{j \in V_D \mid (i,j) \in \mathcal{E}\}$, be the set of destination nodes which have arcs with origin node i; and
$\delta^-_j = \{i \in V_O \mid (i,j) \in \mathcal{E}\}$, be the set of origin nodes which have arcs with destination node j.

With this notation we define pure and generalized transportation problems as follows:

[PTR] Pure Transportation Problem:

$$\text{Minimize} \quad F(x) \tag{1}$$

Subject to :

$$\sum_{j \in \delta^+_i} x_{ij} = s_i, \quad \forall\, i \in V_O, \tag{2}$$

$$\sum_{i \in \delta^-_j} x_{ij} = d_j, \quad \forall\, j \in V_D, \tag{3}$$

$$0 \le x_{ij} \le u_{ij}, \quad \forall\, (i,j) \in \mathcal{E}. \tag{4}$$

[GTR] Generalized Transportation Problem:

$$\text{Minimize} \quad F(x) \tag{5}$$

Subject to :

$$\sum_{j \in \delta^+_i} x_{ij} \le s_i, \quad \forall i \in V_O, \tag{6}$$

$$\sum_{i \in \delta^-_j} m_{ij} x_{ij} = d_j, \quad \forall j \in V_D, \tag{7}$$

$$0 \le x_{ij} \le u_{ij}, \quad \forall (i,j) \in \mathcal{E}. \tag{8}$$

The objective function $F : \Re^n \to \Re$ may take one of the following two forms:

[E] Entropy:

$$F(x) = \sum_{(i,j) \in \mathcal{E}} x_{ij} \left[\ln \left(\frac{x_{ij}}{a_{ij}} \right) - 1 \right], \tag{9}$$

where ln is the natural logarithm and $\{a_{ij}\}$ are given positive real numbers.

[Q] Quadratic:

$$F(x) = \sum_{(i,j) \in \mathcal{E}} \left[\frac{1}{2} w_{ij} x^2_{ij} + c_{ij} x_{ij} \right], \tag{10}$$

where $\{w_{ij}\}$ and $\{c_{ij}\}$ are given positive real numbers.

The following standing assumptions are made for both [PTR] and [GTR].

ASSUMPTION 1. The constraint sets (2)–(4) and (6)–(8) are feasible. (The Gale–Hoffman circulation conditions on the problem data $\mathcal{E}$, s, d, m, and u that guarantee feasibility of the constraint set are given in Ford and Fulkerson [12, Chap. 11].)

ASSUMPTION 2. The transportation graph is connected. (Otherwise, the problem could be partitioned into its disconnected components and each solved separately.)

ASSUMPTION 3. $s_i > 0$ for all $i \in V_O$ and $d_j > 0$ for all $j \in V_D$. (If these conditions are violated for some index i or j, then all the flows on arcs incident to the offending node can be set to zero and the relevant constraint removed from the problem.)

In order to develop the row-action algorithms, we need to express the problems in compact matrix notation. Let S be the $m_O \times n$ matrix:

$$S = \left[\begin{array}{c|c|c|c} \alpha_{11}\ \alpha_{12} \cdots \alpha_{1m_D} & \alpha_{21}\ \alpha_{22} \cdots \alpha_{2m_D} & \cdots & \\ & & & \\ & & & \alpha_{m_O1}\ \alpha_{m_O2} \cdots \alpha_{m_Om_D} \end{array}\right],$$

with entries α_{ij} given by

$$\alpha_{ij} = \begin{cases} 1, & \text{if } j \in \delta_i^+, \\ 0, & \text{otherwise,} \end{cases} \tag{11}$$

and the remaining entries of S all zeros. D is the $m_D \times n$ matrix :

$$D = \left[\begin{array}{c|c|c|c} \beta_{11} & \beta_{21} & \cdots & \beta_{m_O1} \\ \quad \beta_{12} & \quad \beta_{22} & & \quad \beta_{m_O2} \\ \quad\quad \cdot & \quad\quad \cdot & & \quad\quad \cdot \\ \quad\quad\quad \beta_{1m_D} & \quad\quad\quad \beta_{2m_D} & & \quad\quad\quad \beta_{m_Om_D} \end{array}\right]$$

with entries β_{ij} given by

$$\beta_{ij} = \begin{cases} m_{ij}, & \text{if } i \in \delta_j^-, \\ 0, & \text{otherwise,} \end{cases} \tag{12}$$

and the remaining entries of D all zeros.

Finally, let

$$\Phi = \begin{bmatrix} S \\ D \\ I \end{bmatrix},$$

where I is the $n \times n$ identity matrix, and $\phi^l = (\phi_t^l) = (\phi_{lt})$, for all $t \in \langle n \rangle$, denotes the lth column of Φ^T, the transpose of Φ. Also, let

$$\gamma = \begin{bmatrix} -\infty \\ d \\ 0 \end{bmatrix} \quad \text{and} \quad \delta = \begin{bmatrix} s \\ d \\ u \end{bmatrix}.$$

Then equations (6)–(8) can be written in the form

$$\gamma \leq \Phi x \leq \delta. \tag{13}$$

The relationship between the algebraic formulation (5)–(8) and the compact matrix notation is made precise by imposing a lexicographic ordering of the arcs, i.e., $t = (i-1)m_O + j$. Hence, x_t and u_t indicate x_{ij} and u_{ij}, respectively, for some arc $(i,j) \in \mathcal{E}$. Finally, we denote by $z \in \Re^{m_O+m_D+n}$ the vector of dual variables:

$$z = \begin{pmatrix} \pi^O \\ \pi^D \\ r \end{pmatrix}.$$

2.2. The row-action framework. In this section we summarize the general row-action algorithm for interval convex programming of Censor and Lent [7]. The specialized transportation algorithms will be derived later from this general framework. We will start with some preliminary discussion. Let $F : \Lambda \subseteq \Re^n \to \Re$ and let $\Omega \neq \emptyset$ be an open convex set such that its closure $\bar{\Omega} \subseteq \Lambda$. The set Ω is called the *zone* of F if F is strictly convex and continuous on $\bar{\Omega}$, continuously differentiable on Ω, and if F is a Bregman function over Ω (see [7]).

Let $D(x,y) = F(x) - F(y) - \langle \nabla F(y), x-y \rangle$, and let $H(a,b)$ be the hyperplane $H(a,b) = \{x \in \Re^n \mid \langle a,x \rangle = b\}$. The *D-projection* (or *Bregman projection*) of a point y onto $H(a,b)$ is defined by

$$P_{H(a,b)}(y) = \arg \min_{x \in H(a,b) \cap \bar{\Omega}} D(x,y). \tag{14}$$

A function F, which belongs to the family of *Bregman functions* as characterized by Censor and Lent [7], has the *zone consistency property* with respect to the hyperplane $H(a,b)$ if the D-projection of every $y \in \Omega$ onto $H(a,b)$ is also in Ω. If a function is zone-consistent with respect to $H(a,b)$, then it can be shown [7, Lemma 3.1] that the D-projection of y onto $H(a,b)$ is the point x given by the unique solution of the system of equations in x and β:

$$\nabla F(x) = \nabla F(y) + \beta \cdot a, \tag{15}$$

$$\langle a,x \rangle = b. \tag{16}$$

The unique real number β is known as the *Bregman parameter*. It can be interpreted as the stepsize that maximizes the dual functional along the dual price coordinate for equation $\langle a,x \rangle = b$. (See, for example, [23] or [24].)

Both the quadratic and entropy are Bregman functions [7]. It is also easy to verify that, under Assumptions 1–3, both functions have the *strong zone consistency* property with respect to the hyperplanes $H(\phi^l, \gamma_l)$ and $H(\phi^l, \delta_l)$ for all rows of the constraint matrix. Hence we can apply the following general iterative scheme.

Algorithm 2.1. General row-action algorithm.

Step 0: (Initialization). $k \leftarrow 0$. Get $z^0 \in \Omega$ and $x^0 \in \Re^n$ such that

$$\nabla F(x^0) = -\Phi^T z^0. \tag{17}$$

Step 1: (Iterative step over equality constraints). Choose a row index $l(k)$ from the equality constraints and solve the following system for $x^{k+1/2}$ and β_k:

$$\nabla F(x^{k+1/2}) = \nabla F(x^k) + \beta_k \phi^{l(k)} \tag{18}$$

$$z^{k+1/2} = z^k - \beta_k e^{l(k)}, \tag{19}$$

where β_k is the Bregman parameter associated with the D-projection of x^k on the hyperplane $H(\phi^{l(k)}, \gamma_{l(k)})$. $\{l(k)\}$ is the control sequence of the algorithm, henceforth abbreviated as $l = l(k)$. $e^l \in \Re^{m_O+m_D+n}$ is the lth standard basis vector having 1 in the lth coordinate and zeros elsewhere.

Step 2: (Iterative step over interval constraints). Choose a row index $l(k)$ from the interval constraints. Calculate the Bregman parameters Γ_k and Δ_k associated with the D-projection of $x^{k+1/2}$ on the hyperplanes specified when the left and right inequalities, respectively, of the interval constraints hold with equality (i.e., $x^{k+1/2}$ is projected on the hyperplanes $H(\phi^{l(k)}, \gamma_{l(k)})$ and $H(\phi^{l(k)}, \delta_{l(k)})$, respectively). Using the operation "mid" (i.e. if $x \le y \le z$, then $\text{mid}\{x, y, z\} = y$), update x^{k+1} and z^{k+1} as follows:

$$\beta_k = \text{mid}\{z_{l(k)}^{k+1/2}, \Gamma_k, \Delta_k\}. \tag{20}$$

$$\nabla F(x^{k+1}) = \nabla F(x^{k+1/2}) + \beta_k \phi^{l(k)} \tag{21}$$

$$z^{k+1} = z^{k+1/2} - \beta_k e^{l(k)}. \tag{22}$$

Step 3: Let $k \leftarrow k+1$, and return to Step 1.

2.3. Algorithms for the generalized network problems. Consider now the generalized network problem. Constraints (6) and (8) are inequality constraints, and the iterative step can be obtained from Step 2 of the general Algorithm 2.1. Constraints (7) are equality constraints, and the iterative step is obtained by applying Step 1 of the general row-action algorithm. We first iterate over constraints (6), then over (7), and finally project on the bounds, i.e., constraints (8). Of course, any other *almost cyclic control sequence* (see, e.g., [7]) will do, but we have found the sequence proposed here slightly more efficient in practice than other sequences.

2.3.1. Entropy optimization algorithm. We now formulate the iterative algorithm for solving generalized networks with entropy objective function. The iterative step for the entropy objective function (see (18)–(19) or (21)–(22)) is the following:

$$\begin{cases} x_t^{k+1} = x_t^k \exp(\beta_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ z^{k+1} = z^k - \beta_k e^l. \end{cases} \tag{23}$$

For $l = 1, 2, 3, \cdots, m_O$ the parameter β_k is obtained from Step 2 of Algorithm 2.1 as:

$$\beta_k = \min\{(\pi_l^O)^k, p_k\}, \tag{24}$$

where p_k is obtained by solving

$$\begin{cases} y_t = x_t^k \exp(p_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ \langle y, \phi^l \rangle = s_l. \end{cases} \tag{25}$$

Substitute the first equation above into the second, and use the facts that ϕ^l is the ith column of S^T with values $\alpha_{ij} \in \{0,1\}$, as defined in (11), and that $t = (i-1)m_O + j$ is the lexicographic ordering of x_{ij} for $j \in \delta_i^+$. We get

$$\exp(p_k) = \frac{s_i}{\sum_{j \in \delta_i^+} x_{ij}^k}. \tag{26}$$

This value of p_k goes into (24) to get β_k, which is then used by (23) to complete the iterative step.

For $l = m_O + 1, m_O + 2, m_O + 3, \cdots, m_O + m_D$, the parameter β_k is obtained from Step 1 of Algorithm 2.1 by solving the system:

$$\begin{cases} y_t = x_t^k \exp(\beta_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ \langle y, \phi^l \rangle = d_l. \end{cases} \tag{27}$$

Substituting the first equation into the second and using the fact that ϕ^l is the $i \doteq (l - m_O)$th column of D^T with values β_{ij} as given in (12), and $t = (i-1)m_O + j$ is the lexicographic ordering of x_{ij} for $i \in \delta_j^-$, we get:

$$\sum_{i \in \delta_j^-} m_{ij} x_{ij}^k \exp(\beta_k m_{ij}) = d_j. \tag{28}$$

Now denote $\omega_k = \exp(\beta_k)$ and rewrite (28) as a nonlinear equation in ω_k:

$$\psi(\omega_k) \doteq \sum_{i \in \delta_j^-} m_{ij} x_{ij}^k \omega_k^{m_{ij}} - d_j = 0. \tag{29}$$

The existence of a solution to this equation is guaranteed by the feasibility Assumption 1. However, solving (29) for ω_k requires an iterative procedure (e.g., Newton's algorithm). An approximate solution of (29) in closed form can be obtained by taking one secant step as follows: Consider the line through $(0, -d_j)$ and $(1, \psi(1))$ instead of the graph of $\psi(\omega_k)$. This line intersects the ω_k-axis at

$$\hat{\omega}_k = \frac{d_j}{\sum_{i \in \delta_j^-} m_{ij} x_{ij}^k}, \tag{30}$$

and we use this value of $\hat{\omega}_k$ as an approximation to the solution of (29). It has been shown (see [8]) that using this approximation preserves asymptotic convergence of the algorithm. The value $\hat{\omega}_k$ is used to set $\beta_k = \ln \hat{\omega}_k$ and in turn to complete the iterative step in (23).

Finally, to avoid the exponentiation terms in stating the algorithm, we use the exponents of the dual prices, which we denote by $(\overline{\pi}_i^O)$, $(\overline{\pi}_j^D)$, and $(\overline{r}_{ij})$, and use the fact that

$$\ln \min(\exp x, \exp y) = \min(x, y).$$

The algorithm also permits user-chosen relaxation parameters $\{\lambda_k\}$ that may vary between successive iterations as the algorithm iterates on different constraints. To

preserve asymptotic convergence, it is required that $0 < \epsilon \leq \lambda_k \leq 1$. We use λ to indicate the appropriate use of relaxation parameters with the understanding that the numerical value of λ may vary as explained above. The entropy optimization algorithm for generalized transportation problems is stated as follows.

Algorithm 2.2. Entropy optimization for generalized transportation problems.

Step 0: (Initialization). Set $k \leftarrow 0$. Get $x^0 > 0, (\overline{\pi}^O)^0, (\overline{\pi}^D)^0, \overline{r}^0$ such that:

$$x_{ij}^0 = \frac{a_{ij}}{(\overline{\pi}_i^O)^0((\overline{\pi}_j^D)^0)^{m_{ij}}\overline{r}_{ij}^0}. \tag{31}$$

Step 1: (Iterative step over constraint set (6)). Pick a relaxation parameter λ and, for $i = 1, 2, 3, \cdots, m_O$, calculate:

$$\rho_i^k = \left(\frac{s_i}{\sum_{j\in\delta_i^+} x_{ij}^k}\right)^\lambda, \tag{32}$$

$$\Delta\rho_i^k \doteq \min\{(\overline{\pi}_i^O)^k, \rho_i^k\}, \tag{33}$$

$$x_{ij}^k \leftarrow x_{ij}^k \Delta\rho_i^k, \qquad j \in \delta_i^+, \tag{34}$$

$$(\overline{\pi}_i^O)^{k+1} = \frac{(\overline{\pi}_i^O)^k}{\Delta\rho_i^k}. \tag{35}$$

Step 2: (Iterative step over constraint set (7)). Pick a relaxation parameter λ and, for $j = 1, 2, 3, \cdots, m_D$, calculate:

$$\sigma_j^k = \left(\frac{d_j}{\sum_{i\in\delta_j^-} m_{ij}x_{ij}^k}\right)^\lambda, \tag{36}$$

$$x_{ij}^k \leftarrow x_{ij}^k(\sigma_j^k)^{m_{ij}}, \qquad i \in \delta_j^-, \tag{37}$$

$$(\overline{\pi}_j^D)^{k+1} = \frac{(\overline{\pi}_j^D)^k}{\sigma_j^k}. \tag{38}$$

Step 3: (Iterative step over constraint set (8)). Pick a relaxation parameter λ and, for all $(i, j) \in \mathcal{E}$, calculate:

$$\Delta_{ij}^k \doteq \min\left(\overline{r}_{ij}^k, \left(\frac{u_{ij}}{x_{ij}^k}\right)^\lambda\right), \tag{39}$$

$$x_{ij}^{k+1} = x_{ij}^k \Delta_{ij}^k, \tag{40}$$

$$\overline{r}_{ij}^{k+1} = \frac{\overline{r}_{ij}^k}{\Delta_{ij}^k}. \tag{41}$$

Step 4: Replace $k \leftarrow k + 1$ and return to Step 1.

2.3.2. Quadratic optimization algorithm. Consider now the generalized network transportation problem with a quadratic objective function. The iterative step

for this algorithm is derived again from Algorithm 2.1. It can be written (see (18)–(19) or (20)–(21)) in the form:

$$\begin{cases} x_t^{k+1} = x_t^k + (1/w_t)(\beta_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ z^{k+1} = z^k - \beta_k e^l. \end{cases} \tag{42}$$

For $l = 1, 2, 3, \cdots, m_O$, the parameter β_k is

$$\beta_k = \min\{(\pi_l^O)^k, p_k\} \tag{43}$$

where p_k is obtained by solving

$$\begin{cases} y_t = x_t^k + (1/w_t)(p_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ \langle y, \phi^l \rangle = s_l. \end{cases} \tag{44}$$

Substitute the first equation above into the second, and use the facts that ϕ^l is a column of S^T with values $\alpha_{ij} \in \{0, 1\}$, as defined in (11), and that $t = (i-1)m_O + j$ is the lexicographic ordering of x_{ij} for $j \in \delta_i^+$. We get

$$p_k = \frac{1}{\sum_{j \in \delta_i^+} 1/w_{ij}} \left[s_i - \sum_{j \in \delta_i^+} x_{ij}^k \right]. \tag{45}$$

This value of p_k goes into (43) to get β_k, which in turn goes into (42) to complete the iterative step.

For $l = m_O + 1, m_O + 2, m_O + 3, \cdots, m_O + m_D$, the parameter β_k is obtained by solving

$$\begin{cases} y_t = x_t^k + (1/w_t)(\beta_k \phi_t^l), & t = 1, 2, 3, \cdots, n, \\ \langle y, \phi^l \rangle = d_l. \end{cases} \tag{46}$$

Substitute the first equation into the second, and then using the fact that ϕ^l is the $i \doteq (l - m_O)$th column of D^T with values β_{ij} as given in (12), and $t = (i-1)m_O + j$ is the lexicographic ordering of x_{ij} for $i \in \delta_j^-$, we get:

$$\beta_k = \frac{1}{\sum_{i \in \delta_j^-} m_{ij}^2 / w_{ij}} \left[d_j - \sum_{i \in \delta_j^-} m_{ij} x_{ij}^k \right]. \tag{47}$$

This value of β_k is used in (42) to complete the iterative step. It is worth noting that this iterative step for the quadratic optimization problem performs an exact minimization along the jth coordinate of the dual vector π^D. Contrast this to the computation of β_k for the entropy optimization algorithm (28) where a closed form solution could not be obtained and an approximation was used instead.

Once more, relaxation parameters are allowed and they may take different values for each update that involves a single constraint. To guarantee the asymptotic convergence of the algorithm they must be confined to the interval $0 < \epsilon \leq \lambda_k \leq 2 - \epsilon$, for some positive ϵ. In the interest of keeping the algorithm description simple we just write λ whenever a relaxation parameter should appear.

Algorithm 2.3. Quadratic optimization for generalized transportation problems.

Step 0: (Initialization). Set $k \leftarrow 0$. Get $x^0, (\pi^O)^0, (\pi^D)^0, r^0$ such that:

$$x_{ij}^0 = -\frac{1}{w_{ij}} \left[c_{ij} + (\pi_i^O)^0 + m_{ij}(\pi_j^D)^0 + r_{ij}^0 \right]. \tag{48}$$

Step 1: (Iterative step over constraint set (6)). Pick a relaxation parameter λ and, for $i = 1, 2, 3, \cdots, m_O$, calculate:

$$\rho_i^k = \frac{\lambda}{\sum_{j \in \delta_i^+} 1/w_{ij}} \left[s_i - \sum_{j \in \delta_i^+} x_{ij}^k \right], \tag{49}$$

$$\Delta\rho_i^k = \min\{(\pi_i^O)^k, \rho_i^k\}, \tag{50}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \frac{\Delta\rho_i^k}{w_{ij}}, \qquad j \in \delta_i^+, \tag{51}$$

$$(\pi_i^O)^{k+1} = (\pi_i^O)^k - \Delta\rho_i^k. \tag{52}$$

Step 2: (Iterative step over constraint set (7)). Pick a relaxation parameter λ and, for $j = 1, 2, 3, \cdots, m_D$, calculate:

$$\sigma_j^k = \frac{\lambda}{\sum_{i \in \delta_j^-} m_{ij}^2/w_{ij}} \left[d_j - \sum_{i \in \delta_j^-} m_{ij} x_{ij}^k \right], \tag{53}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \sigma_j^k \frac{m_{ij}}{w_{ij}}, \qquad i \in \delta_j^-, \tag{54}$$

$$(\pi_j^D)^{k+1} = (\pi_j^D)^k - \sigma_j^k. \tag{55}$$

Step 3: (Iterative step over constraint set (8)). Pick a relaxation parameter λ and, for all $(i, j) \in \mathcal{E}$, calculate:

$$\Delta_{ij}^k \doteq \text{ mid } \{r_{ij}^k, \lambda w_{ij}(u_{ij} - x_{ij}^k), -\lambda w_{ij} x_{ij}^k\}, \tag{56}$$

$$x_{ij}^{k+1} = x_{ij}^k + \frac{\Delta_{ij}^k}{w_{ij}}, \tag{57}$$

$$r_{ij}^{k+1} = r_{ij}^k - \Delta_{ij}^k. \tag{58}$$

Step 4: Replace $k \leftarrow k + 1$ and return to Step 1.

2.4. Algorithms for pure network problems. The algorithms for the pure network problem [PTR] can now be obtained as special cases of the generalized network algorithms of the previous section. Just observe that the nonzero entries of the matrix D for the pure network problem are all equal to one. Hence, we can substitute $m_{ij} = 1$ in Algorithms 2.2 and 2.3 to obtain algorithms for the pure network problems. It is worth pointing out, however, that for the entropy optimization algorithm on pure network problems, the value of $\hat{\omega}_k$ given in (30) is an exact solution to the nonlinear equation (28). This is likely to be an important practical advantage for the solution of pure networks.

The algorithm for entropy optimization over the pure transportation constraints is the following.

Algorithm 2.4. Entropy optimization for pure transportation problems.

Step 0: (Initialization). Set $k \leftarrow 0$ and get $x^0 > 0, (\overline{\pi}^O)^0 \geq 0, (\overline{\pi}^D)^0 \geq 0$, and $\overline{r}^0 \geq 0$ such that:

$$x_{ij}^0 = \frac{a_{ij}}{(\overline{\pi}_i^O)^0 (\overline{\pi}_j^D)^0 \overline{r}_{ij}^0}. \tag{59}$$

Step 1: (Iterative step over constraint set (2)). Pick a relaxation parameter λ and, for $i = 1, 2, 3, \cdots, m_O$, calculate:

$$\rho_i^k = \left(\frac{s_i}{\sum_{j \in \delta_i^+} x_{ij}^k} \right)^\lambda, \tag{60}$$

$$x_{ij}^k \leftarrow x_{ij}^k \rho_i^k, \qquad j \in \delta_i^+, \tag{61}$$

$$(\overline{\pi}_i^O)^{k+1} = \frac{(\overline{\pi}_i^O)^k}{\rho_i^k}. \tag{62}$$

Step 2: (Iterative step over constraint set (3)). Pick a relaxation parameter λ and, for $j = 1, 2, 3, \cdots, m_D$, calculate:

$$\sigma_j^k = \left(\frac{d_j}{\sum_{i \in \delta_j^-} x_{ij}^k} \right)^\lambda, \tag{63}$$

$$x_{ij}^k \leftarrow x_{ij}^k \sigma_j^k, \qquad i \in \delta_j^-, \tag{64}$$

$$(\overline{\pi}_j^D)^{k+1} = \frac{(\overline{\pi}_j^D)^k}{\sigma_j^k}. \tag{65}$$

Step 3: (Iterative step over constraint set (4)). Pick a relaxation parameter λ and, for all $(i, j) \in \mathcal{E}$, calculate:

$$\Delta_{ij}^k \doteq \min \left(\overline{r}_{ij}^k, \left(\frac{u_{ij}}{x_{ij}^k} \right)^\lambda \right), \tag{66}$$

$$x_{ij}^{k+1} = x_{ij}^k \Delta_{ij}^k, \tag{67}$$

$$\overline{r}_{ij}^{k+1} = \frac{\overline{r}_{ij}^k}{\Delta_{ij}^k}. \tag{68}$$

Step 4: Replace $k \leftarrow k + 1$ and return to Step 1.

Note that $\overline{\pi}^O$ and $\overline{\pi}^D$ are irrelevant for the equality-constrained problem and, therefore, need not be iterated upon. In the case where the starting point is taken as $x_{ij}^0 = a_{ij}$, $u_{ij} = \infty$, for all $(i, j) \in \mathcal{E}$, and the price updating steps are ignored, Algorithm 2.4 coincides with the RAS algorithm for matrix-balancing problems (see, e.g., [22]). Note that the constraints $x_{ij} \geq 0$ in (4) are automatically enforced due to the multiplicative nature of the iterative steps (13) and (19), the positive initialization of x and Assumption 3.

The algorithm for quadratic optimization of pure network problems is obtained as a special case of Algorithm 2.3 by setting $m_{ij} = 1$. It can be stated as follows.

Algorithm 2.5. Quadratic optimization for pure transportation problems.

Step 0: (Initialization). Set $k \leftarrow 0$. Get $x^0, (\pi^O)^0, (\pi^D)^0, r^0$ such that:

$$x_{ij}^0 = -\frac{1}{w_{ij}} \left[c_{ij} + (\pi_i^O)^0 + (\pi_j^D)^0 + r_{ij}^0 \right]. \tag{69}$$

Step 1: (Iterative step over constraint set (2)). Pick a relaxation parameter λ and, for $i = 1, 2, 3, \cdots, m_O$, calculate:

$$\rho_i^k = \frac{\lambda}{\sum_{j \in \delta_i^+} 1/w_{ij}} \left[s_i - \sum_{j \in \delta_i^+} x_{ij}^k \right], \tag{70}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \frac{\rho_i^k}{w_{ij}}, \quad j \in \delta_i^+, \tag{71}$$

$$(\pi_i^O)^{k+1} = (\pi_i^O)^k - \rho_i^k. \tag{72}$$

Step 2: (Iterative step over constraint set (3)). Pick a relaxation parameter λ and, for $j = 1, 2, 3, \cdots, m_D$, calculate:

$$\sigma_j^k = \frac{\lambda}{\sum_{i \in \delta_j^-} 1/w_{ij}} \left[d_j - \sum_{i \in \delta_j^-} x_{ij}^k \right], \tag{73}$$

$$x_{ij}^k \leftarrow x_{ij}^k + \frac{\sigma_j^k}{w_{ij}}, \qquad i \in \delta_j^-, \tag{74}$$

$$(\pi_j^D)^{k+1} = (\pi_j^D)^k - \sigma_j^k. \tag{75}$$

Step 3: (Iterative step over constraint set (4)). Pick a relaxation parameter λ and, for all $(i, j) \in \mathcal{E}$, calculate:

$$\Delta_{ij}^k \doteq \text{ mid } \{r_{ij}^k, \lambda w_{ij}(u_{ij} - x_{ij}^k), -\lambda w_{ij} x_{ij}^k\}, \tag{76}$$

$$x_{ij}^{k+1} = x_{ij}^k + \frac{\Delta_{ij}^k}{w_{ij}}, \tag{77}$$

$$r_{ij}^{k+1} = r_{ij}^k - \Delta_{ij}^k. \tag{78}$$

Step 4: Replace $k \leftarrow k + 1$ and return to Step 1.

2.5. The Connection Machine CM-2. We now briefly introduce the characteristics of the Connection Machine (model CM-2) [14] that are relevant to our parallel implementations. The Connection Machine is a fine-grain SIMD (i.e., single instruction stream, multiple data stream) system. Its basic hardware component is an integrated circuit with 16 processing elements (PEs) and a *router* that handles general communication. A fully configured CM has 4,096 chips for a total of 65,536 PEs. The 4,096 chips are interconnected as a 12-dimensional hypercube. Each processor is equipped with local memory of 8Kbytes, and for each cluster of 32 PEs a floating-point accelerator handles floating-point arithmetic.

Operations by the PEs are under the control of a *microcontroller* that broadcasts instructions from a front-end computer (FE) simultaneously to all the elements for

execution. A flag register at every PE allows for no-operations; i.e., an instruction received from the microcontroller is executed if the flag is set, and ignored otherwise.

Parallel computations on the CM are in the form of a single operation executed on multiple copies of the problem data. All processors execute identical operations, each one operating on data stored in its local memory, accessing data residing in the memory of other PEs, or receiving data from the front end. This mode of computation is termed *data level parallelism* in contradistinction to *control level parallelism*, whereby multiple processors execute their own control sequence, operating either on local or shared data.

To achieve high performance with data level parallelism one needs a large number of processors. The CM provides the mechanism of *virtual processors* (VPs) that allows one PE to operate in a serial fashion on multiple copies of data. VPs are specified by slicing the local memory of each PE into equal segments and allowing the physical processor to loop over all slices. The number of segments is called the VP *ratio* (i.e., ratio of virtual to physical PEs). Looping by the PE over all the memory slices is executed, in the worst case, in linear time. The set of virtual processors associated with each element of a data set is called a VP *set.*

The CM supports two addressing mechanisms for communication. The *send* address is used for general purpose communications via the routers. The NEWS address describes the position of a VP in an n-dimensional grid that optimizes communication performance. The *send* address indicates the location of the PE (hypercube address) that supports a specific VP and the relative address of the VP in the VP set that is currently active. NEWS address is an n-tuple of coordinates which specifies the relative position of a VP in an n-dimensional Cartesian-grid geometry. A *geometry* is an abstract description of such an n-dimensional grid. Once a geometry is associated with the currently active VP set, a relative addressing mechanism is established among the processors in the VP set. Each processor has a relative position in the n-dimensional geometry and NEWS allows the communication across the North, East, West, and South neighbors of each processor, and enables the execution of operations along the axes of the geometry. Such operations are efficient since the n-dimensional geometry can be mapped onto the underlying hypercube in such a way that adjacent VPs are mapped onto vertices of the hypercube connected with a direct link. This mapping of an n-dimensional mesh on a hypercube is achieved through a Gray coding (see, e.g., [4, p. 50]).

The algorithm was implemented using C/Paris. Paris is the lowest level protocol by which the actions of the data processors of the CM are controlled by the front end. Before invoking Paris instructions from a program the user has to specify the VP set, create a geometry, and associate the VP set with the geometry. Thus a communications mechanism is established (along both *send* and NEWS addresses). Paris instructions—parallel primitives—can then be invoked to execute operations along some axis of the geometry (using NEWS addresses), to operate on an individual processor using *send* addresses, or to translate NEWS to *send* addresses for general interprocessor communication or communication with the front end.

Parallel primitives that are relevant to our implementation are the *scans* and *spreads* of Blelloch [5]. The $\otimes$-scan primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \cdots, x_n\}$ and produces another sequence $\{y_0, y_1, \cdots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i$. For example, *add-scan* takes as an argument a parallel variable (i.e., a variable with its ith element residing in a memory field of the ith VP) and returns at VP i the value of the parallel variable summed over

Processing Element	=	0	1	2	3	4	5	6	7	8	9
X	=	5	1	3	4	3	9	2	6	1	0
Segment Bits (Sb)	=	1	0	1	0	0	0	1	0	0	1
$Y = add\text{-}scan(X, Sb)$	=	0	5	6	3	7	10	19	2	8	9
$copy\text{-}scan(Y, Sb)$	=	0	0	0	6	6	6	6	19	19	19
$reverse\text{-}copy\text{-}scan(Y, Sb)$	=	6	6	19	19	19	19	9	9	9	9

FIG. 1. *An example of the segmented* add-scan *and* copy-scan *primitives.*

$j = 0, \cdots, i$. User options allow the scan to apply only to preceding processors (e.g., sum over $j = 0, \cdots, i-1$) or to perform the scan in reverse. The $\otimes$-spread primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \cdots, x_n\}$ and produces another sequence $\{y_0, y_1, \cdots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_n$. For example, *add-spread* takes as an argument a parallel variable residing in the memories of n active data processors and returns at VP i the value of the parallel variable summed over $j = 0, \cdots, n$.

Another variation of the scan primitives allows their operation within *segments* of a parallel variable. These primitives are denoted as *segmented-$\otimes$-scan.* They take as arguments a parallel variable and a set of segment bits which specify a partitioning of the VP set into contiguous segments. Segment bits have a one at the starting location of a new segment and zeros elsewhere. A *segmented-$\otimes$-scan* operation restarts at the beginning of every segment. Figure 1 illustrates the use of *segmented-add-scan* and *segmented-copy-scan.* When processors are configured as a NEWS grid, scans within rows or columns are special cases of segmented scans called *grid-scans.*

3. Data level parallel implementations. We discuss in this section the implementation of Algorithms 2.4 and 2.5 for the pure network problem on the Connection Machine CM-2. The implementations can be easily extended to Algorithms 2.2 and 2.3 in order to handle generalized networks.

The Connection Machine is based on very simple processing elements, each running at approximately 7 MHz of clock cycle. Furthermore, the processing elements are bit-serial and floating-point calculations are performed by a floating-point accelerator that is shared among 32 processing elements. Hence, implementations of an algorithm that do not fully exploit the large number of PEs are likely to be very inefficient. We will show in this section that the algorithms of this paper can be implemented in a way that takes advantage of the architecture. As a result, the implementations are very efficient and solve the large test problems within a few seconds of solution time.

The key to the implementations is the layout of the problem data on some user-specified configuration of the processing elements. The algorithm is then executed by multiple PEs operating on local data, and communicating with each other through the prespecified configuration grid. We first give details of dense implementations of both algorithms, and then discuss the data structures required for sparse implementations. The layout of the PEs for both implementations is borrowed from [27], but the memory of each PE is partitioned into different fields as required for the entropy or quadratic optimization algorithms, respectively.

3.1. Dense implementation of the entropy optimization algorithm. The CM-2 is configured as a two-dimensional NEWS grid. One dimension is set equal to the number of origin nodes m_O rounded up to the nearest integer that is a power

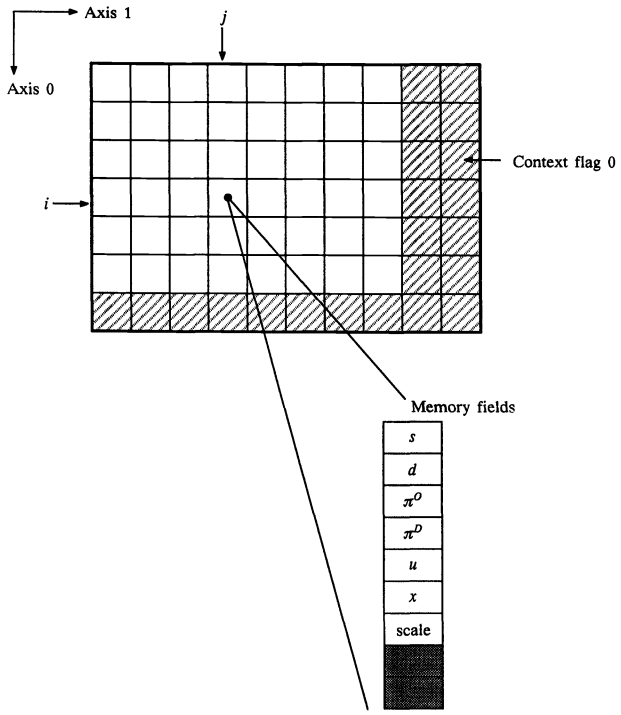


FIG. 2. *NEWS grid configuration and memory partitioning for dense implementation of the entropy optimization Algorithm* 2.4.

of two, and the other dimension is set equal to the number of destination nodes m_D rounded up in the same fashion. Virtual processors outside the $m_O \times m_D$ grid have their context flags set to 0 and hence do not participate in the computations. The memory of virtual processor with NEWS address (i, j) is partitioned into the following data fields:

1. Supply and demand, s and d.
2. *pii*, *pij*, and r to store the dual prices, $\overline{\pi}^O$, $\overline{\pi}^D$, and $\overline{r}$.
3. *upp* to store the upper bound u.
4. Current iterate x.
5. A scaling factor *scale* used to store ρ, σ, or intermediate results.

The configuration of the CM-2 as a two-dimensional grid and the memory partitioning of each virtual processor are illustrated in Fig. 2. One iteration of entropy optimization Algorithm 2.4 is executed as follows:

Step 1: A *spread-with-add* operation along the 1-axis of the geometry computes the partial sums $\sum_{j \in \delta_i^+} x_{ij}^k$ for each origin node (i.e., each row of the grid). This result is spread to the *scale* memory fields of all VPs in the same row.
A division operation of local fields s by *scale* computes the scaling factor (ρ_i^k of (60)), which again is stored in memory field *scale*. The scaling factor

```
/* Scaling of origin nodes */
   CM_spread_with_f_add_1L(scale  , x, axis_news[1], S, E);
   CM_f_sub_mult_1L(scale, s      , scale , odeg   , S, E);
   CM_f_mult_add_1L(x     , invcfl , scale , x      , S, E);
   CM_f_subtract_2_1L(pii, scale                   , S, E);

/* Scaling of destination nodes */
   CM_spread_with_f_add_1L(scale  , x, axis_news[0], S, E);
   CM_f_sub_mult_1L(scale, d      , scale , ideg   , S, E);
   CM_f_mult_add_1L(x     , invcfl , scale , x      , S, E);
   CM_f_subtract_2_1L(pij, scale                   , S, E);

/* Scaling of bounds */
   CM_f_multiply_3_1L(scale, cfl   , x             , S, E); /* Lower bound */
   CM_f_negate_1_1L  (scale                        , S, E);
   CM_f_max_2_1L     (scale, r                     , S, E);
   CM_f_sub_mult_1L  (scr  , upp   , x,cfl         , S, E); /* Upper bound */
   CM_f_min_2_1L     (scale, scr                   , S, E);
   CM_f_mult_add_1L  (x    , invcfl, scale, x      , S, E); /* Scale       */
   CM_f_subtract_2_1L(r    , scale                 , S, E);
```

FIG. 3. *C/Paris implementation of entropy optimization Algorithm* 2.4.

```
/* Scaling of origin nodes */
CM_spread_with_f_add_1L(scale  , x, axis_news[1], S, E);
CM_f_divinto_2_1L      (scale  , s              , S, E);
CM_f_multiply_2_1L     (x      , scale          , S, E);
CM_f_divinto_2_1L      (pii    , scale          , S, E);

/* Scaling of destination nodes */
CM_spread_with_f_add_1L(scale  , x, axis_news[0], S, E);
CM_f_divinto_2_1L      (scale  , d              , S, E);
CM_f_multiply_2_1L     (x      , scale          , S, E);
CM_f_divinto_2_1L      (pij    , scale          , S, E);

/* Scaling of bounds */
CM_f_divide_3_1L  (scale, upp  , x              , S, E);
CM_f_min_2_1L     (scale, r                     , S, E);
CM_f_multiply_2_1L(x    , scale                 , S, E);
CM_f_divinto_2_1L (r    , scale                 , S, E);
```

FIG. 4. *C/Paris implementation of quadratic optimization Algorithm* 2.5.

updates the local field x by multiplication and the local field *pii* by division.

Step 2: This step is similar to Step 1 with the exception that the *spread-with-add* operation is executed along the 0-axis. The local fields used in the calculations are those pertaining to destination nodes (i.e., d and pij).

Step 3: The ratio of local fields *upp* over x is stored in *scale*. A *min* operation computes the correction term Δ_{ij}^k of equation (66), which is once again stored in *scale* and is used in turn to update the local value of x and r.

Figure 3 gives the C/Paris implementation of one complete iteration of the algorithm.

3.2. Dense implementation of the quadratic optimization algorithm. The quadratic optimization algorithm is implemented using the configuration of the CM-2 given in the previous section. The memory of each virtual processor with NEWS

address (i, j) is partitioned into the following data fields:

1. Supply and demand, s and d.
2. *pii*, *pij*, and r to store the dual prices π^O, π^D, and r.
3. *upp* to store the upper bound u. (The lower bound l is assumed to be equal to zero and hence it is treated as a constant.)
4. Current iterate x.
5. A scaling factor *scale* used to store ρ, σ, or intermediate results.
6. Constants: two fields *ideg* and *odeg* hold the terms

$$\frac{1}{\sum_{i\in\delta_j^-} 1/w_{ij}} \quad \text{and} \quad \frac{1}{\sum_{j\in\delta_i^+} 1/w_{ij}},$$

respectively. $cf1$ holds w_{ij} and $invcf1$ holds $1/w_{ij}$.

With this configuration of the CM-2, one iteration of the algorithm is executed as follows:

Step 1: A *spread-with-add* operation along the 1-axis of the geometry computes the partial sums $\sum_{j\in\delta_i^+} x_{ij}^k$ for each origin node (i.e., each row of the grid). This result is spread to the *scale* memory fields of all VPs in the same row.
The *sub-mult* instruction is an optimized implementation of the operation $a(x-b)$. It is used to compute the scaling factor ρ (70). The scaling factor updates the local field x by addition and the local field *pii* by subtraction ((71) and (72), respectively).

Step 2: This step is similar to Step 1 with the exception that the *spread-with-add* operation is executed along the 0-axis. The local fields used in the calculations are those pertaining to destination nodes (i.e., d and *pij*) and the scaling factor σ.

Step 3: A combination of *min* and *max* operations computes the *mid* of equation (76) as follows:

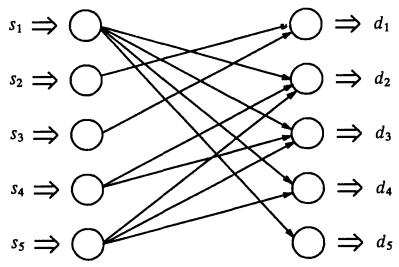
$$\text{mid}\{a, b, c\} = \max\{a, \min\{b, c\}\}.$$

(It is easy to verify that the expression on the right is identical to the expression on the left when $a \le c$. To evaluate equation (76) set $a = -\lambda w_{ij} x_{ij}^k$ and $c = \lambda w_{ij}(u_{ij} - x_{ij})$.) The result of the *mid* operation is then used to update the local field x by a *mult-add* operation (77) and local field r (78).

Figure 4 gives the C/Paris implementation of one complete iteration of the algorithm.

3.3. Sparse data structures. The solution of dense transportation problems has a natural mapping onto the two-dimensional NEWS grid. It is possible to solve sparse transportation problems using the same data representation of §§3.1 and 3.2 and turning off VPs corresponding to arcs that are not present in the transportation graph. In essence, a sparse problem is represented as a dense problem, and operations of the algorithm simply ignore zero values. It is possible, however, to represent the problem as a sparse transportation graph. This representation will reduce the memory requirements, thus allowing the solution of much bigger problems.

To represent sparse problems, we use a simplification of the data structures of Zenios and Lasken [25], that were also used in the implementation of matrix-balancing algorithms in Zenios [27]. While the configuration of the CM-2 as a one-dimensional NEWS grid is identical to the one used in the matrix-balancing applications, the data fields allocated in the memory of the virtual processors are different for each algorithm.



NEWS address of VP	1	2	3	4	5	6	7	8	9	10	11	12
s-orig	1	0	0	0	1	1	1	0	1	0	0	1
x-orig	x_{12}	x_{13}	x_{14}	x_{15}	x_{21}	x_{31}	x_{42}	x_{43}	x_{52}	x_{53}	x_{54}	
s	s_1	s_1	s_1	s_1	s_2	s_3	s_4	s_4	s_5	s_5	s_5	
p-orig	3	6	9	11	1	2	4	7	5	8	10	
p-dest	5	6	1	7	9	2	8	10	3	11	4	
d	d_1	d_1	d_2	d_2	d_2	d_3	d_3	d_3	d_4	d_4	d_5	
x-dest	x_{21}	x_{31}	x_{12}	x_{42}	x_{52}	x_{13}	x_{43}	x_{53}	x_{14}	x_{54}	x_{15}	
s-dest	1	0	1	0	0	1	0	0	1	0	1	1

FIG. 5. *Representation of a sparse transportation problem.*

In this sparse implementation we use VPs in the one-dimensional NEWS grid to operate only on arcs that are present in the transportation graph. Every arc is stored twice: once in a format that allows efficient execution of iterations over the origin nodes and once in a format that allows efficient execution of the iterations over the destination nodes. This scheme has some redundancy in the use of virtual processors, but allows the efficient use of *segmented-scan* operations along the axis of the NEWS grid, as explained later.

Figure 5 illustrates the sparse representation of a small problem. The memory of each VP is partitioned into the following data fields:

1. Field *s-orig* of segment bits. It is used to partition the VP set into segments such that VPs in the ith segment correspond to arcs (i, j) for all $j \in \delta_i^+$.
2. Field *x-orig* that holds the value of the current iterate x_{ij}^k. The entries are sorted by origin node: arcs outgoing from node i belong to the same segment as delimited by *s-orig*.
3. Field s holds the supply values s_i. All VPs that correspond to the same node (i.e., are in the same row segment) hold identical values in this field.
4. A second field of segment bits *s-dest* partitions the VP set into segments, such that VPs in the jth segment correspond to arcs (i, j) for all $i \in \delta_j^-$.
5. Field *x-dest* holds the value of the current iterate x_{ij}^k sorted by destination node: arcs incoming into node j belong to the same segment as delimited by *s-dest*. This field provides redundant information, since the nonzero entries

```
/* Get send address coordinates for permutation arrays */

CM_deposit_news_coordinate_1L (vec_geometry     , send_orig          ,
                       vec_axis_news[axis]      , p_orig , LINT      );

CM_deposit_news_coordinate_1L (vec_geometry     , send_dest          ,
                       vec_axis_news[axis]      , p_dest , LINT      );


/* Scaling of origin nodes */

CM_scan_with_f_add_1L(scale       , x_orig       , vec_axis_news[axis],
                      S_REAL      , E_REAL       , CM_upward          ,
                      CM_inclusive, CM_start_bit, s_orig              );

CM_f_divinto_2_1L    (scale       , s            , S_REAL , E_REAL    );

CM_scan_with_copy_1L (scale       , scale        , vec_axis_news[axis],
                      REAL        , CM_downward  , CM_inclusive       ,
                      CM_start_bit, cs_orig                           );

CM_f_multiply_2_1L   (x_orig      , scale        , S_REAL , E_REAL    );

/* Copy x from origin to destination */
CM_send_1L           (x_dest      , send_orig    , x_orig , REAL , 0  );


/* Scaling of destination nodes */

CM_scan_with_f_add_1L(scale       , x_dest       , vec_axis_news[axis],
                      S_REAL      , E_REAL       , CM_upward          ,
                      CM_inclusive, CM_start_bit, s_dest              );

CM_f_divinto_2_1L    (scale       , d            , S_REAL , E_REAL    );

CM_scan_with_copy_1L (scale       , scale        , vec_axis_news[axis],
                      REAL        , CM_downward  , CM_inclusive       ,
                      CM_start_bit, s_dest                            );

CM_f_multiply_2_1L   (x_dest      , scale        , S_REAL , E_REAL    );

/* Copy x from destination to origin */
CM_send_1L           (x_orig      , send_dest    , x_dest , REAL , 0  );
```

FIG. 6. *Sparse implementation of entropy optimization Algorithm* 2.4 *in* C/*Paris.*

have already been stored by origin node in field *x-orig*.

6. Field d holds the demand values d_j. All VPs that correspond to the same column (i.e., are in the same segment as delimited by *s-dest*) hold identical values.
7. Field *scale* is used as a register for temporary storage of intermediate results.
8. Two fields *p-orig* and *p-dest* hold pointers to map the nonzero entries from the origin field *x-orig* to the destination field *x-dest* and vice versa.

With this representation of sparse problems, one iteration of the entropy Algorithm 2.4 is executed as shown in Fig. 6. First the permutation arrays *p-orig* and *p-dest*, which are pointers to the NEWS addresses, are translated into *send* addresses. This allows transferring of data between the row and column representation using the routers. A *segmented-add-scan* operation on field *x-orig* over the origin node segments computes the sum of outgoing flows from each node. This partial sum is then

used to divide the supply field s, thus computing the scaling factors. Note that only the last processor in each segment has the sum of all the outgoing flows and hence only the scaling factor of the last VP is the correct one. A *reverse segmented-copy-scan* is used to copy the correct scaling factor to all VPs in the same segment. Finally, each VP proceeds to multiply its copy of the current iterate $x\text{-}orig$ with its local copy of the scaling factor.

Before the algorithm can proceed with the iteration over destination nodes the current iterate $x\text{-}orig$, just scaled following the origin node operations, have to be copied into field $x\text{-}dest$. This is a rearrangement of the contents of the memory field $x\text{-}orig$ according to the sparsity structure of the graph. The NEWS addresses stored in the pointer field $p\text{-}orig$ have already been converted into *send* addresses and router communications are used to copy the nonzero entries of the matrix from the row field $x\text{-}orig$ to the column field $x\text{-}dest$.

4. Computational results. We implemented the algorithms for pure network problems on a Connection Machine CM-2, as explained in the previous section. The algorithms were implemented in C/Paris under release B5.1, microcode 5110, using single precision arithmetic (i.e., 23 digits in the mantissa). The programs were compiled with the optimization flags -O and object code was optimized for the CM-2 (-cm2 option of the compiler). All experiments were carried out on the CM-2 at the North-East Parallel Architectures Center (NPAC) at Syracuse University, Syracuse, NY. The particular configuration has 32K PEs and each PE has 8Kbytes of local memory. The front end is a VAX8700 running Unix BSD4.2. All times are in seconds, exclusive input and output, and were obtained using the Paris instructions CM-start-timer and CM-stop-timer. The relaxation parameter is set equal to $\lambda_k = \lambda = 1$ in all experiments.

4.1. Test problems. Test problems for the entropy optimization algorithm were randomly generated as follows: For a given dimension of the transportation problem, we specify the density factor of the transportation graph, the range of flows on each arc, and the range of cost coefficients. For every origin-destination pair, the generator will either add an arc with flow uniformly distributed in the prespecified range, or will determine that no arc exists. Presence or absence of arcs is determined by the density factor. The flows on all the arcs incident to a node are added up to determine the supply or demand at each node. The flows are then discarded, and the coefficients $\{a_{ij}\}$ are generated for all arcs from the prespecified cost range. The characteristics of the entropy optimization problems are summarized in Table 1. Coefficients $\{a_{ij}\}$ are uniformly distributed in the range 1 to 10. The entropy test problems are uncapacitated.

The quadratic optimization problems were generated by a modified version of NETGEN of Klingman, Napier, and Stutz [17]. The code was modified to generate coefficients for a quadratic function of the form: $\frac{1}{2}wx^2 + cx$. The coefficients w are randomly generated from the interval [5, 10], c is generated from the interval [1, 1000], and supply/demand vectors have all their components in the interval [1, 1000]. We use the test problems generated by Chajakis and Zenios [9] to evaluate dual relaxation algorithms for strictly convex network problems on a shared memory multiprocessor. Hence, a direct comparison of the two approaches is possible. The parameters specified for the NETGEN code are identical to those used by Tseng [23] in his experiments with dual coordinate ascent methods for problems with strictly

TABLE 1
Test problem characteristics: Entropy optimization problems.

Problem	Supply/Demand range	Number of nodes	Number of arcs
ENT1	0 - 1238	500 × 500	5016
ENT2	0 - 1178	750 × 750	7355
ENT3	0 - 1301	1000 ×1000	9965
ENT4	0 - 1021	1250 ×1250	12318
ENT5	0 - 1009	500 × 500	9950
ENT6	0 - 1043	750 × 750	14453
ENT7	0 - 974	1000 ×1000	19600
ENT8	0 - 1021	1250 ×1250	24239
ENT9	0 - 1646	1000 ×1000	39064
ENT10	0 - 1680	1000 ×1000	97752
ENT11	0 - 2088	1000 ×1000	360219
ENT12	0 - 1669	1000 ×1000	601131

TABLE 2
Test problem characteristics: Quadratic optimization problems.

Problem	Nodes	Arcs
TSENG1	500 × 500	5063
TSENG2	750 × 750	7611
TSENG3	1000× 1000	10126
TSENG4	1250× 1250	12665
TSENG5	500× 500	10051
TSENG6	750× 750	15086
TSENG7	1000× 1000	20134
TSENG8	1250× 1250	25153

convex costs.[1] Hence, the results of the two methods can again be compared. The characteristics of the quadratic optimization problems are summarized in Table 2.

4.2. Solving the entropy optimization problems. The entropy optimization algorithm was initialized with $\overline{\pi}^O = 1, \overline{\pi}^D = 1, \overline{r} = 1$, and $x_{ij}^0 = a_{ij}$ for all $(i,j) \in \mathcal{E}$. The termination criteria are $\mid 1 - \rho_i \mid \leq 10^{-6}$ and $\mid 1 - \sigma_j \mid \leq 10^{-6}$. With this termination criteria, the final error defined by

$$\max_{i,j} \left\{ \left| \frac{s_i - \sum_{j \in \delta_i^+} x_{ij}}{s_i} \right|, \left| \frac{d_j - \sum_{i \in \delta_j^-} x_{ij}}{d_j} \right| \right\}$$

is always less than 10^{-6}. Testing the termination criteria is an expensive step, since it involves communication among all the processors of the grid. Hence, we test for convergence only every fifth iteration. Table 3 summarizes the performance of the algorithm on the entropy test problems. Under the column "No. of Iterations" we report the number of iterative steps of the parallel algorithm over *all* the nodes of the graph. For example, 20 iterations on problem ENT4 correspond to scaling and updating 20 × 2500 = 50000 nodes.

To obtain an estimate of the solution time when solving the test problems on a fully configured CM-2 with 64K PEs we solved one of the test problems (ENT1) using

[1] Numerical results appeared in the technical report by Tseng, but are not included in the published article. The results in Table 5 were obtained from the technical report [23].

TABLE 3
Solution of the entropy optimization test problems using the dense C/Paris implementation of Algorithm 2.4 on a 32K CM-2.

Problem	VP ratio	No. of iterations	CM time	Total time	Estimated CM time on 64K CM-2
ENT1	8	15	0.09	0.10	0.05
ENT2	32	15	0.29	0.30	0.15
ENT3	32	20	0.39	0.39	0.19
ENT4	128	20	2.00	2.00	0.75
ENT5	8	10	0.06	0.07	0.03
ENT6	32	10	0.20	0.20	0.10
ENT7	32	10	0.20	0.20	0.10
ENT8	128	10	1.00	1.00	0.38
ENT9	32	10	0.21	0.22	0.11
ENT10	32	5	0.11	0.13	0.06
ENT11	32	5	0.10	0.11	0.05
ENT12	32	5	0.11	0.11	0.06

different VP ratios (ranging from 8 to 256). The degradation in performance of the algorithm as the VP ratio increases is recorded in Fig. 7. From this figure we compare the solution time for solving a given problem under one VP ratio on the 32K CM-2, with the solution time for the same problem on the 64K CM-2 under a different VP ratio (equal to one half of the VP ratio on the 32K CM-2). This information has been provided for all the test problems in Table 3. Finally, we mention that the algorithm achieves a computing rate of approximately 600 MFLOPS on the 64K CM-2 when solving 1000 × 1000 dense problems. The FLOPS rate is estimated by multiplying the total number of Paris floating-point instructions by the number of VPs and dividing by the CM time. The useful computing rate decreases proportionally to the sparsity of the problem. For example, when solving a 1000 × 1000 problem with 500,000 arcs (i.e., 50 percent sparse), the algorithm achieves a computing rate of 300 MFLOPS since only half of the one million VPs are active.

4.3. Solving the quadratic optimization problems. The quadratic optimization algorithm was initialized with $\pi^O = 0, \pi^D = 0, r = 0$, and $x_{ij}^0 = -(c_{ij}/w_{ij})$ for all $(i,j) \in \mathcal{E}$. The termination criteria are:

$$\max_{i,j} \left\{ \left| s_i - \sum_{j \in \delta_i^+} x_{ij} \right|, \left| d_j - \sum_{i \in \delta_j^-} x_{ij} \right| \right\} \leq \epsilon_1 \frac{1}{n} \left\{ \sum_i s_i + \sum_j d_j \right\}.$$

The algorithm terminates following a projection on the bounds, hence the bounds are satisfied exactly. Termination conditions are tested every fifth iteration. Table 4 summarizes the performance of the dense implementation of the algorithm on the quadratic test problems for termination tolerances ϵ_1 equal to 10^{-3}, 10^{-4}, and 10^{-6}, respectively. It is quite remarkable that the algorithm can achieve a very tight tolerance (i.e., $\epsilon_1 = 10^{-6}$), even when implemented using single precision arithmetic. We observe, however, a significant increase in the number of iterations with the increase of the desired level of accuracy.

The TSENG test problems have very large upper bounds. While several variables are at the lower bound (i.e., 0) at the optimal solution, none of the upper bounds are active. We modified TSENG5 to force some of the upper bounds to be active at the

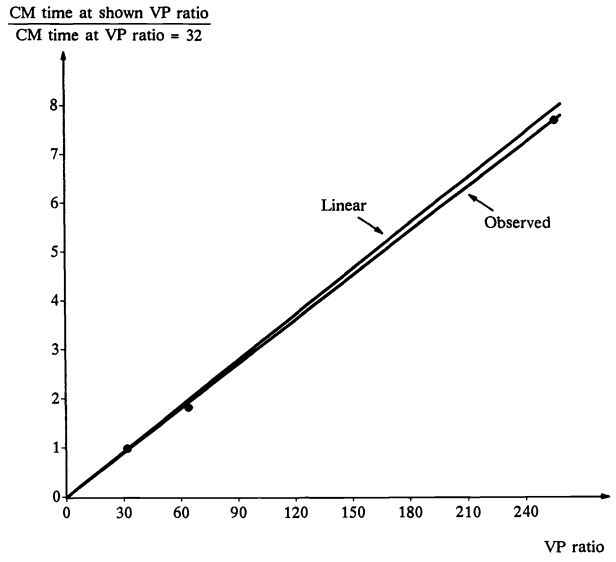


FIG. 7. *Degradation in performance of the* C/*Paris implementation of the entropy optimization Algorithm* 2.4 *with varying* VP *ratios.*

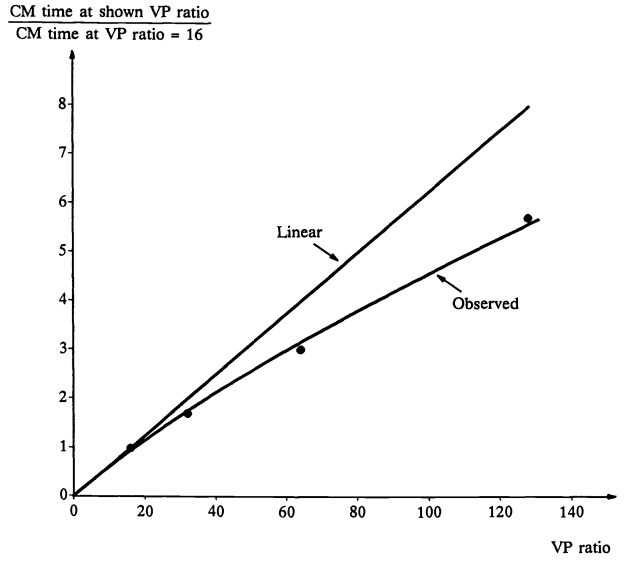


FIG. 8. *Degradation in performance of the* C/*Paris implementation of the quadratic optimization Algorithm* 2.5 *with varying* VP *ratios.*

TABLE 4
Solution of the quadratic optimization test problems with different levels of accuracy using the dense C/Paris implementation of Algorithm 2.5 on a 32K CM-2.

Problem	VP ratio	$\epsilon_1 = 10^{-3}$		$\epsilon_1 = 10^{-4}$		$\epsilon_1 = 10^{-6}$	
		Itns.	CM time	Itns.	CM time	Itns.	CM time
TSENG1	8	250	1.91	360	3.06	850	6.75
TSENG2	32	300	6.52	3000	66.98	5000	121.58
TSENG3	32	250	5.43	530	12.01	1200	29.82
TSENG4	128	250	18.64	400	30.53	4050	353.61
TSENG5	8	200	1.53	650	5.86	1000	7.84
TSENG6	32	400	8.71	890	22.00	1700	38.97
TSENG7	32	250	5.43	790	17.51	1250	27.95
TSENG8	128	200	15.05	850	66.27	1150	93.48

optimal solution, by reducing the upper bounds. When the number of active bounds would increase from 0 to 15 percent of the total number of variables, we observed a very small increase in the number of iterations. As the number of variables with active bounds was increased to 20 percent the test problem would become infeasible.

To obtain an estimate of the solution time when solving the test problems on a 64K CM-2, we conducted the experiment outlined in §4.2. Solving problem TSENG1 with different VP ratios, we obtain the curve of Fig. 8. The results of the figure are used in estimating the solution time on a 64K CM-2 when the VP ratio is half of the ratio needed on a 32K CM-2: the CM times reported in Table 4 for the 32K CM-2 should be divided by 1.7 to obtain estimated run times on the 64K CM-2.

The algorithm achieves a peak computing rate of approximately 1.75 GFLOPS when solving dense 1000 × 1000 problems on the 64K CM-2. The FLOPS rate is estimated by multiplying the total number of Paris floating-point instructions by the number of VPs and dividing by the CM time. The useful computing rate decreases proportionally to the sparsity of the problem. For example, when solving a 1000 × 1000 problem with 500,000 arcs (i.e., 50 percent sparse) the algorithm achieves a computing rate of 800 MFLOPS since only half of the one million VPs are active. An optimal implementation is also possible (see McKenna and Zenios [18]), which runs at 3 GFLOPS. Hence, all the times reported in Table 4 would be reduced by a factor of 0.58 if they were obtained with the optimal implementation. However, the latter implementation was carried out in CMIS microcode and direct timings are not reported here, since we want to emphasize the efficiency of implementations in a high-level language.

4.4. Comparing the dense with the sparse implementations. The bipartite transportation problems match naturally with the two-dimensional NEWS grid of the Connection Machine. However, for the very sparse problems we are solving here, the NEWS grid representation could be inefficient. Especially, for the bigger TSENG4 and TSENG8 problems the algorithm is using a NEWS grid of dimension 2048 × 2048 for a total of 4 million virtual processors. For TSENG4 only 12665 VPs are active and for TSENG8 only 25153. In contrast, the sparse implementation uses only $m_O + m_D + 2n$ virtual processors, rounded up to the nearest integer power of 2. For TSENG4 and TSENG8 this corresponds to 32K and 64K virtual processors, respectively. Hence, for a given machine size the sparse implementation runs at a much lower VP ratio than the dense implementation. However, the sparse implementation uses a general router communication step which is substantially more expensive than

TABLE 5
Comparing the dense code on a 32K CM-2 *with the sparse code on the* 16K CM-2. *Solution times in* CM *seconds.*

Problem	Dense code		Sparse code	
	VP ratio	CM time	VP ratio	CM time
TSENG1	8	1.91	1	1.55
TSENG2	32	6.52	2	2.45
TSENG3	32	5.43	2	2.03
TSENG4	128	18.64	2	2.06
TSENG5	8	1.53	2	1.63
TSENG6	32	8.71	2	3.43
TSENG7	32	5.43	4	3.45
TSENG8	128	15.05	4	2.85

the NEWS grid operations of the dense implementation.

To compare the dense with the sparse implementations we solved the test problems at a tolerance of $\epsilon_1 = 10^{-3}$. The sparse code was executed on a 16K CM-2, and run at VP ratios ranging from 1 to 4. The dense code was executed on the 32K CM-2, and run at VP ratios ranging from 8 to 128. Results are summarized in Table 5. We observe that the sparse code is significantly faster for most problems. TSENG5 is the only exception to this observation. For this problem the dense code runs at a VP ratio of 8 and the sparse code at a VP ratio of 2. For this difference in VP ratios the efficiency of the NEWS communications in the dense code balances the lower VP ratio of the sparse code.

4.5. Comparisons with other methods. We compare the results from Table 5 with the work reported in Chajakis and Zenios [9] and Tseng [23] (see footnote 1). Chajakis and Zenios use a dual relaxation algorithm, specialized for vector and parallel computing on an Alliant FX/8. Their most efficient implementation on eight processors is compared to the results with our quadratic programming algorithm. Tseng is using a dual relaxation algorithm on a microVAX–II workstation. He implements two versions of the algorithm, one with exact line searches (LNS1) and one with a simple line search procedure that is easy to implement but is inexact (LNS2). In Table 6 we summarize the solution times with all three methods. (Termination tolerance $\epsilon_1 = 10^{-3}$ was used by all authors.)

Finally, we generated and solved some very large quadratic transportation problems according to the specifications given in the paper by Nagurney, Kim, and Robinson [20, p. 58, Table 1, Growth Factor=1]. These problems were solved by Nagurney, Kim, and Robinson using a row-equilibration algorithm on an IBM 3090 vector supercomputer, with a termination tolerance $\epsilon_1 = 10^{-2}$. A smaller problem has also been solved by Klincewicz [16] on the IBM 3090 using an exact Newton algorithm. (Klincewicz's termination criteria are not directly comparable to ours.) The results from all three approaches are summarized in Table 7.

5. Conclusions. Several conclusions can be drawn from the results of this study. First, the row-action algorithm is a simple version of the relaxation algorithms of Tseng and Chajakis and Zenios. As such, it can be implemented on a fine-grain SIMD architecture like the Connection Machine, while the relaxation algorithm with line search can be implemented very effectively on a coarse-grain MIMD architecture like the Alliant FX/8. We observe from the results of Table 6 that the massively parallel implementation of the simple algorithm can significantly outperform

TABLE 6

Solution times of the quadratic optimization algorithm on the CM-2 *compared with the results reported in Chajakis and Zenios* [9] *on an Alliant* FX/8, *and the results of Tseng* [23] *on a microVAX-II.*

Problem	CM-2		Alliant FX/8		MicroVax II	
	16K PEs	32K PEs	1 proc.	8 proc.	LNS1	LNS2
TSENG1	1.55	1.55	15.63	3.01	27.07	55.86
TSENG2	2.45	1.47	44.03	14.47	42.40	83.59
TSENG3	2.03	1.22	38.26	7.31	58.18	130.50
TSENG4	2.06	1.23	43.14	9.24	70.67	127.34
TSENG5	1.63	0.98	27.60	4.33	54.40	139.34
TSENG6	3.43	2.05	39.51	6.61	78.92	188.49
TSENG7	3.45	2.07	52.74	9.15	106.24	259.24
TSENG8	2.85	1.71	71.77	11.41	135.62	307.63

TABLE 7

Comparing the row-action algorithm on the CM-2 *with the equilibration algorithm of Nagurney, Kim, and Robinson* [20] *and the exact Newton algorithm of Klincewicz* [16] *on the* IBM 3090 *for the solution of dense quadratic optimization problems. Times in* CM *and* CPU *hrs:min:sec, respectively.*

Problem size	Row-action CM-2	Equilibration IBM 3090	Exact Newton IBM 3090
500 × 500	0:00:18	0:01:09	0:30:00
1000 × 1000	0:00:22	0:08:03	NA
2000 × 2000	0:00:47	1:03:43	NA

the coarse-grain parallel implementation of the more complex algorithm. Secondly, the results of Table 7 quote two of the most recent papers on the solution of very large quadratic transportation problems using vector supercomputers, like the IBM 3090. The massively parallel implementation of the simple row-action algorithms once more outperforms the more complex algorithms implemented on vector supercomputers. Thirdly, we observe from both tables that the massively parallel algorithms are insensitive to the problem size: TSENG8 is solved in 1.71 seconds, which is slightly higher than the 1.55 seconds required for the solution of the smaller TSENG1. The increase in solution time with problem size observed in Table 7 is much slower with the massively parallel algorithm, than with the equilibration algorithm on the vector supercomputer.

In empirical studies of the sort reported in the second half of this paper, one wants to compare the best algorithms implemented on different computer architectures. Of course, such a task is very difficult since it is by no means clear which algorithm is "best" for these applications. Our work has shown that the simple row-action algorithms on a massively parallel architecture can outperform by a large margin serial algorithms on vector supercomputers (like the IBM 3090) and competing parallel algorithms (i.e., relaxation) on coarse-grain parallel architectures.

Of course, any comparison among competing computer architectures should take into account the relative price/performance ratio. It is worth pointing out that the vector supercomputer and the coarse-grain architecture can be viewed as general purpose systems, while it is not clear at present whether the massively parallel SIMD architecture can solve as broad a range of applications. Such issues, although important, are beyond the scope of our study. However, our results provide additional evidence to an increasing body of literature that claims that massively parallel com-

puters with suitably structured numerical algorithms can significantly outperform vector supercomputers and coarse-grain parallel systems.

Acknowledgments. We would like to acknowledge insightful comments from two anonymous referees that led to substantial improvements in our presentation. Access to the Connection Machine CM-2 was made possible through the North-East Parallel Architectures Center (NPAC) of Syracuse University, Syracuse, NY, and Thinking Machines Corporation, Cambridge, MA.

REFERENCES

[1] D. P. AHLFELD, R. S. DEMBO, J. M. MULVEY, AND S. A. ZENIOS, *Nonlinear programming on generalized networks*, ACM Trans. Math. Software, 13 (1987), pp. 350–368.

[2] A. BACHEM AND B. KORTE, *Minimum norm problems over transportation polytopes*, Linear Algebra Appl., 31 (1980), pp. 103–118.

[3] D. P. BERTSEKAS, P. HOSSEIN, AND P. TSENG, *Relaxation methods for network flow problems with convex arc costs*, SIAM J. Control Optim., 25 (1987), pp. 1219–1243.

[4] D. P. BERTSEKAS AND J. N. TSITSIKLIS, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, Englewood Cliffs, NJ, 1989.

[5] G. E. BLELLOCH, *Vector Models for Data-Parallel Computing*, The MIT Press, Cambridge, MA, 1990.

[6] L. M. BREGMAN, *The relaxation method for finding the common point of convex sets and its application to the solution of problems in convex programming*, USSR Comput. Math. and Math. Phys., 7 (1967), pp. 200–217.

[7] Y. CENSOR AND A. LENT, *An iterative row-action method for interval convex programming*, J. Optim. Theory Appl., 34 (1981), pp. 321–353.

[8] Y. CENSOR, A. R. DE PIERRO, T. ELFVING, G. T. HERMAN, AND A. N. IUSEM, *On iterative methods for linearly constrained entropy maximization*, in Numerical Analysis and Mathematical Modelling, Vol. 24, A. Wakulicz, ed., Banach Center Publications, PWN-Polish Scientific Publishers, Warsaw, Poland, 1990, pp. 145–163.

[9] E. D. CHAJAKIS AND S. A. ZENIOS, *Synchronous and asynchronous implementations of relaxation algorithms for nonlinear network optimization*, Parallel Comput., (1991), to appear.

[10] R. S. DEMBO, J. M. MULVEY, AND S. A. ZENIOS, *Large scale nonlinear network models and their application*, Oper. Res., 37 (1989), pp. 353–372.

[11] T. ELFVING, *An algorithm for maximum entropy image reconstruction from noisy data*, Math. Comput. Modelling, 12 (1989), pp. 729–745.

[12] L. R. FORD, JR. AND D. R. FULKERSON, *Flows on Networks*, Princeton University Press, Princeton, NJ, 1962.

[13] F. HARRIGAN AND I. BUCHANAN, *A quadratic programming approach to input-output estimation and simulation*, J. Regional Science, 24 (1984), pp. 339–358.

[14] W. D. HILLIS, *The Connection Machine*, The MIT Press, Cambridge, MA, 1985.

[15] L. V. KANTOROVICH, *On the translocation of masses*, Compt. Rend. Acad. Sci. U.R.S.S., 37 (1942), pp. 366–422.

[16] J. G. KLINCEWICZ, *Implementing an exact Newton method for separable convex transportation problems*, Networks, 19 (1989), pp. 95–105.

[17] D. KLINGMAN, A. NAPIER, AND J. STUTZ, NETGEN—*a program for generating large-scale (un)capacitated assignment, transportation, and minimum cost flow network problems*, Management Science, 20 (1974), pp. 814–822.

[18] M. MCKENNA AND S. A. ZENIOS, *An optimal parallel implementation of a quadratic transportation algorithm*, in Fourth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 357–363.

[19] D. L. MILLER, J. F. PEKNEY, AND G. L. THOMPSON, *Solution of large dense transportation problems using a parallel primal algorithm*, Management Science Research Report No. 546, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[20] A. NAGURNEY, D-S. KIM, AND A. G. ROBINSON, *Serial and parallel equilibration of large-scale constrained matrix problems with application to the social and economic sciences*, Internat. J. Supercomput. Appl., 4 (1990), pp. 49–71.

[21] A. OHUCHI AND I. KAJI, *Lagrangian dual coordinatewise maximization algorithm for network transportation problems with quadratic costs*, Networks, 14 (1984), pp. 515–530.

[22] M. H. SCHNEIDER AND S. A. ZENIOS, *A comparative study of algorithms for matrix balancing*, Oper. Res., 38 (1990), pp. 439–455.

[23] P. TSENG, *Dual ascent methods for problems with strictly convex costs and linear constraints: a unified approach*, SIAM J. Control Optim., 28 (1990), pp. 214–242; also appeared as Tech. Report LIDS-P-1792, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, July 1988.

[24] P. TSENG AND D. P. BERTSEKAS, *Relaxation methods for problems with strictly convex costs and linear inequality constraints*, Math. Oper. Res., to appear.

[25] S. A. ZENIOS AND R. A. LASKEN, *Nonlinear network optimization on a massively parallel Connection Machine*, Ann. Oper. Res., 14 (1988), pp. 147–165.

[26] S. A. ZENIOS AND J. M. MULVEY, *Relaxation techniques for strictly convex network problems*, Ann. Oper. Res., 5 (1986), pp. 517–538.

[27] S. A. ZENIOS, *Matrix balancing on a massively parallel Connection Machine*, ORSA J. Comput., 2 (1990), pp. 112–125.

[28] G. T. HERMAN AND A. LENT, *A family of iterative quadratic optimization algorithms for pairs of inequalities, with application in diagnostic radiology*, Math. Programming Stud., 9 (1978), pp. 15–29.

[29] C. HILDRETH, *A quadratic programming procedure*, Naval Res. Logist. Quart., 4 (1957), pp. 79–85. Erratum, ibid., p. 361.

[30] A. LENT AND Y. CENSOR, *Extensions of Hildreth's row-action method for quadratic programming*, SIAM J. Control Optim., 18 (1980), pp. 444–454.

SIAM J. OPTIMIZATION
Vol. 1, No. 3, pp. 401–423, August 1991

006

ON DUAL CONVERGENCE AND THE RATE OF PRIMAL CONVERGENCE OF BREGMAN'S CONVEX PROGRAMMING METHOD*

ALFREDO N. IUSEM†

Abstract. Bregman's method is an iterative algorithm for solving optimization problems with convex objective and linear inequality constraints. It generates two sequences: one, the primal one, is known to converge to the solution of the problem. Under the assumption of smoothness of the objective function at the solution, it is proved that the other sequence, the dual one, converges to a solution of the dual problem, and that the rate of convergence of the primal sequence is at least linear.

Key words. convex programming, iterative algorithms, entropy maximization, large and sparse matrices

AMS(MOS) subject classification. 10C30

1. Introduction.

Bregman's method is an iterative algorithm for solving optimization problems with convex objective and linear inequality constraints. It generates two sequences: one of so-called primal variables, which, under adequate assumptions, converges to a solution of the problem, and another of so-called dual variables, associated with the constraints, which play the role of Kuhn–Tucker multipliers.

Constraints are used one at a time, and the constraint matrix is not changed along the algorithm, thus preserving sparsity and making the method useful for problems with a huge number of variables and constraints. Methods with these characteristics are called "row-action" algorithms (see [3]).

The algorithm has been mostly applied to entropy maximization problems, which arise in a large variety of applications; see, e.g., [8], [10], [11], and [16].

Relevance of the method has been enhanced by the fact that some other methods for entropy maximization (like MART, widely used in image reconstruction) have been shown to be particular instances of Bregman's method with a special relaxation strategy (see [4], [5]), and the theory behind Bregman's method has been used to extend convergence results for such methods.

The status of the convergence theory of Bregman's method is the following: under reasonable hypotheses the primal sequence converges to the solution of the problem and it is known that if the dual sequence converges, its limit is a dual solution. With additional conditions on the constraints (like nonnegativity of the matrix), it can be proved that the dual sequence is bounded and that the difference between consecutive iterates tends to zero, but even in such favorable cases the issue of convergence of the dual sequence was until now an open problem.

The question is relevant because in some cases the dual problem may be of interest on its own. Recent studies on the duality of optimization problems involving entropy functionals (see [1]) led to formulations whose duals are problems that can be solved with Bregman's method. In [15] we present an algorithm for maximum likelihood estimation, akin to the Expectation-Maximization algorithm, whose convergence

*Received by the editors December 11, 1989; accepted for publication (in revised form) December 18, 1990. This research was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico grant 301280/86.

†Instituto de Matemática Pura e Aplicada, Rio de Janeiro 22460, Brazil.

results from the fact that the sequence it generates can be seen as the dual sequence of Bregman's method when applied to Burg's entropy.

We prove convergence of the dual sequence with the additional hypothesis of smoothness of the objective function at the solution. This condition is essential, as shown by a counterexample presented in §8. In order to prove dual convergence, we establish first that the rate of convergence of the primal sequence under that hypothesis is linear, an important result on its own. Our proof is based on the approximation of the objective function by a quadratic one near the solution and the linear convergence rate of the quadratic version of Bregman's method, known as Hildreth's (see [14]).

In this paper $\| \ \|$ and $\langle \ , \ \rangle$ denote the Euclidean norm and inner product, ∇ and ∇^2 are the gradient and Hessian matrix of a function, superindex T denotes transpose, $R^n_{\geq 0}$ is the nonnegative orthant of R^n and e^i is the vector with components $e^i_j = \delta_{ij}$ (Kronecker's delta).

2. Bregman functions and Bregman's algorithm. Consider an open and convex set $S \subset R^n$ with closure $\overline{S}$ and a function $f: \overline{S} \to R$.

Define:

$$D_f: \overline{S} \times S \to R, \quad D_f(x,y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle \tag{1}$$

$$R_1(y,h) = \{x \in \overline{S} : D_f(x,y) \leq h\}$$
$$R_2(x,h) = \{y \in S : D_f(x,y) \leq h\}.$$

DEFINITION 1. A function f is said to be a Bregman function with zone S if and only if

(i) f is continuously differentiable on S.

(ii) f is strictly convex and continuous on $\overline{S}$.

(iii) For every $h \in R$, the level sets $R_1(y,h)$, $R_2(x,h)$ are bounded for every $y \in S$, every $x \in \overline{S}$, respectively.

(iv)
$$y^k \underset{k\to\infty}{\longrightarrow} y^* \in \overline{S} \Rightarrow D_f(y^*, y^k) \underset{k\to\infty}{\longrightarrow} 0.$$

(v) If
$$D_f(x^k, y^k) \underset{k\to\infty}{\longrightarrow} 0, \qquad y^k \underset{k\to\infty}{\longrightarrow} y^* \in S$$
and $\{x^k\}$ is bounded, then
$$x^k \underset{k\to\infty}{\longrightarrow} y^*.$$

D_f is used as a sort of distance, because $D_f(x,y) \geq 0$ and $D_f(x,y) = 0$ if and only if $x = y$.

Given a hyperplane $L = \{x \in R^n : \langle a, x \rangle = \beta\}$ and a point $y \in S$ we say that x is the Bregman projection of y onto L if and only if

$$x = \operatorname*{argmin}_{z \in L \cap \overline{S}} D_f(z,y). \tag{2}$$

Existence and uniqueness of Bregman projections when $L \cap \overline{S} \neq \phi$ are guaranteed by Lemma 2.2 of [7]. By definition, the Bregman projection of a point in S lies in $\overline{S}$. For Bregman's algorithm to work, we need to have such projections in S. So we have the following definition.

DEFINITION 2. A Bregman function f is said to be strongly zone consistent with respect to the hyperplane L if and only if for all $y \in S$ and any hyperplane L' parallel

to L that lies between y and L, $L' \cap S$ is nonempty and the Bregman projection of y onto L' belongs to S.

Take $A \in R^{m\times n}$, $b \in R^m$. Let a^i be the rows of A. Assume $a^i \neq 0$ $(1 \leq i \leq m)$. Let

$$L_i = \{x \in R^n : \langle x, a^i\rangle = b_i\} \qquad (1 \leq i \leq m). \tag{3}$$

We make an additional assumption, concerning both f and A, b:
(vi) f is strongly zone consistent with respect to L_i $(1 \leq i \leq m)$.
Bregman's algorithm is used to solve the problem:

$$\min f(x) \tag{4}$$

$$\text{s.t} \quad Ax \leq b \tag{5}$$

$$x \in \overline{S}. \tag{6}$$

Bregman's algorithm generates two sequences: $\{x^k\} \subset R^n$ and $\{z^k\} \subset R^m$, called the primal and dual sequences, respectively. A general description of the algorithm is the following: at iteration k, pick up an index $i(k)$ in the set $\{1, \cdots, m\} = M$ and compute x^{k+1} as the Bregman projection of x^k onto a hyperplane L' parallel to $L_{i(k)}$ and lying between x^k and $L_{i(k)}$, whose exact location depends upon the value of z^k. Formally the algorithm is as follows:

(a) *Control sequence.* Consider a sequence $i(k)$ $(k = 0, 1, \cdots)$ of elements in M such that there exists an integer r so that for all $i \in M$ and all $k \geq 0$, $i(\ell) = i$ for some ℓ between $k+1$ and $k+r$. Such a sequence is called almost cyclical and r is the constant of almost cyclicality. $i(k)$ indicates the row of A to be used in the kth iteration. Every row is used at least once in any cycle of r consecutive iterations.

(b) *Relaxation.* Consider a sequence of real numbers $\{\alpha_k\} \subset (c, 1]$, for some $c > 0$. The α_k's will be called relaxation parameters.

(c) *Initialization.* Take $z^0 \in R^m_{>0}$, and let x^0 be such that $\nabla f(x^0) = -A^T z^0$.

(d) *Iterative step.* Given $(x^k, z^k) \in R^{n+m}$:

(i) Solve the system:

$$\nabla f(t) - \mu a^{i(k)} = \nabla f(x^k), \tag{7}$$

$$\left\langle t, a^{i(k)}\right\rangle = \alpha_k b_{i(k)} + (1 - \alpha_k)\left\langle x^k, a^{i(k)}\right\rangle \tag{8}$$

in the unknowns t, μ. Let $\mu_k = \mu$ and

$$\gamma_k = \min\{z^k_{i(k)}, \mu_k\}. \tag{9}$$

(ii) Solve the system

$$\nabla f(y) = \nabla f(x^k) + \gamma_k a^{i(k)} \tag{10}$$

in the unknown y.

(iii) Take

$$x^{k+1} = y, \tag{11}$$

$$z^{k+1} = z^k - \gamma_k e^{i(k)}. \tag{12}$$

The properties of Bregman functions and hypothesis (vi) guarantee that both systems have unique solutions. In fact, t is the Bregman projection of x^k onto the hyperplane

$$\overline{L}_k = \left\{x: \left\langle x, a^{i(k)}\right\rangle = \alpha_k b_{i(k)} + (1-\alpha_k)\left\langle x^k, a^{i(k)}\right\rangle\right\}$$

parallel to $L_{i(k)}$ and lying between x^k and $L_{i(k)}$, because $0 \leq \alpha_k \leq 1$. The use of an almost cyclical control sequence is convenient because in some cases the order in which the rows of A are used may change along time. In [7] an algorithm is discussed for a feasible region of the form $b' \leq Ax \leq b$. Each double inequality is taken as a unique constraint, but convergence is proved by showing that the algorithm is equivalent to Bregman's algorithm with $2m$ inequalities and a control sequence with $r = m+1$. If $r = m$ the control is cyclical. Finally we note that strong zone consistence guarantees that $\{x^k\} \subset S$.

Bregman's algorithm was first proposed in [2] and further analyzed in [7] (which incorporates almost cyclical control) and [9] (which adds relaxation).

In principle, each iterative step of the algorithm requires the solution of (7)–(8), a system of $n+1$ equations in $n+1$ unknowns, and, if $\gamma_k \neq \mu_k$, the solution of (10), a system of n equations in n unknowns. Note that these systems are nonlinear unless f is quadratic. Numerical solution of such systems would make the algorithm rather slow. Fortunately, in many cases of interest the situation is not that bad. To begin with, for many separable functions f the system $\nabla f(x) = a$ can be solved in closed form. Such is the case, for instance, when f is Shannon's entropy ($f(x) = \sum_{j=1}^n x_j \log x_j$) or Burg's entropy ($f(x) = -\sum_{j=1}^n \log x_j$). When this happens (7) and (10) are solved immediately and substitution of (7) into (8) gives rise to a single nonlinear equation in the real unknown μ. However, even this single equation can be avoided: under a particular choice of the sequence of relaxation parameters $\{\alpha_k\}$ this equation can be solved in closed form. Such analysis is carried over in [4] for Shannon's entropy. The resulting algorithm, called MART, was already known and widely used in Image Reconstruction from Projections. The fact that it happens to be a particular case of Bregman's method made it possible to obtain more general convergence results. In [5], [6] a similar approach is followed for Burg's entropy, resulting in several new algorithms which employ only closed formulae. Since Bregman's method is a row-action algorithm (see [3]) all these algorithms become preferable to other algorithms for linearly constrained convex programming in cases where A is huge and sparse, since A is not modified at all and therefore there is no fill-in whatsoever. The same applies to the algorithm for Maximum Likelihood Estimation presented in [15].

A complete analysis of (7)–(12) and a proof of convergence of the sequence $\{x^k\}$ can be found in [9]. We will use the following results.

PROPOSITION 1. (i) $\nabla f(x^k) = -A^T z^k$ *for all* k.

(ii) $z^k \geq 0$ *for all* k.

(iii) *There exists* σ_k *such that* $0 \leq \sigma_k \leq \alpha_k$ *and* x^{k+1} *is the Bregman projection of* x^k *onto the hyperplane* $L'_k = \{x : \left\langle x, a^{i(k)}\right\rangle = \sigma_k b_{i(k)} + (1-\sigma_k)\left\langle x^k, a^{i(k)}\right\rangle\}$. L'_k *is parallel to* $L_{i(k)}$ *and lies between* x^k *and* $L_{i(k)}$.

(iv) *The sequence* $\{x^k\}$ *converges to the unique solution* x^* *of* (4)–(6).

(v) *If* $\left\langle a^{i(k)}, x^k\right\rangle < b_{i(k)}$ *then* $\mu_k \geq 0$.

These statements are, respectively, Propositions 4.2, 4.3, 4.4, Theorem 4.1, and Proposition 2.2 of [9].

3. Bregman's algorithm with quadratic objective. Consider a function $\tilde{f}(x) = \xi + q^T x + \frac{1}{2} x^T Q x$, with $\xi \in R$, $q \in R^n$, $Q \in R^{n \times n}$, where Q is symmetric positive-definite. Then $\tilde{f}$ is a Bregman function with

$$D_{\tilde{f}}(x, y) = (x - y)^T Q (x - y)$$

and it is easy to verify the following proposition.

PROPOSITION 2. *The Bregman projection of $x \in R^n$ onto a hyperplane $L = \{x : \langle a, x \rangle = \beta\}$ is the orthogonal projection of x onto L with respect to the inner product $\langle \, , \, \rangle_Q$ defined as $\langle x, y \rangle_Q = x^T Q y$.*

Proof. The proof follows immediately from (13). □

We remark that in this case the Bregman projection y of x is defined by:

$$y = x + \left(\frac{\beta - \langle a, x \rangle}{a^T Q a} \right) Q^{-1} a = x + \left(\frac{\beta - \langle \overline{a}, x \rangle_Q}{\|\overline{a}\|_Q^2} \right) \overline{a}, \tag{13}$$

where $\overline{a} = Q^{-1} a$.

Bregman's algorithm with a quadratic objective is known as Hildreth's algorithm, first presented in [12], and further analyzed in [17] (which incorporates almost cyclical control) and [13] (where a simultaneous, rather than sequential, version is presented).

We end this section with some consequences of Propositions 1 and 2.

LEMMA 1. *If f is quadratic, $\{x^k\}$ is the sequence generated by (7)–(12), $Q = \nabla^2 f(x)$, $\overline{a}^i = Q^{-1} a^i$ $(1 \le i \le m)$, $u \in L_{i(k)}$, and γ_k is as defined in (12), then:*

(i)

$$\|x^{k+1} - u\|_Q \le \|x^k - u\|_Q . \tag{14}$$

(ii)

$$|\gamma_k| \le \frac{1}{\|\overline{a}^{i(k)}\|_Q} \|x^k - u\|_Q . \tag{15}$$

Proof. Let $\overline{x}$ be the orthogonal projection of x^k onto $L_{i(k)}$ with respect to $\langle \, , \, \rangle_Q$. By Propositions 1(iii) and 2, x^{k+1} is in the segment between x^k and $\overline{x}$. The definition and nonexpansiveness of orthogonal projections insure that for all $u \in L_{i(k)}$,

$$\|\overline{x} - u\|_Q \le \|x^k - u\|_Q , \tag{16}$$

$$\|\overline{x} - x^k\|_Q \le \|x^k - u\|_Q . \tag{17}$$

From (12),

$$\nabla f(x^{k+1}) = \gamma_k a^{i(k)} + \nabla f(x^k) \Rightarrow x^{k+1} = x^k + \gamma_k Q^{-1} a^{i(k)} = x^k + \gamma_k \overline{a}^{i(k)} \Rightarrow$$

$$|\gamma_k| = \frac{1}{\|\overline{a}^{i(k)}\|_Q} \|x^{k+1} - x^k\|_Q \le \frac{1}{\|\overline{a}^{i(k)}\|_Q} \|\overline{x} - x^k\|_Q \le \frac{1}{\|\overline{a}^{i(k)}\|_Q} \|x^k - u\|_Q ,$$

where we use (17), (16) and x^{k+1} between x^k and $\overline{x}$. (ii) is proved. (i) results from convexity of $\| \ \|_Q$. □

4. The convergence rate of Hildreth's algorithm. In [14] it is proved that the rate of convergence of Hildreth's algorithm is linear. Since we intend to establish the rate of convergence of Bregman's algorithm through approximation of the function f by a quadratic function in a neighborhood of the solution, we need to examine in detail the results of [14].

A block of r consecutive iterations as defined by (7)–(12) will be called a major iteration (r is the constant of almost cyclicality, i.e., in each major iteration each row of A is used at least once). As could be expected, the linear convergence rate will apply to major iterations. [14] considers the case in which the matrix Q is the identity. More specifically, the problem considered in [14] is:

$$\min \|x - x^0\|^2 \tag{18}$$

$$\text{s.t} \quad \langle a^i, x\rangle \leq b_i \qquad (1 \leq i \leq m). \tag{19}$$

We are interested in problems of the form:

$$\min(x - \hat{x})^T Q(x - \hat{x})$$
$$\text{s.t} \quad \langle a^i, x\rangle \leq b_i,$$

which can be rewritten as:

$$\min \|x - \hat{x}\|_Q^2 \tag{20}$$

$$\text{s.t} \quad \langle \overline{a}^i, x\rangle_Q \leq b_i \qquad (1 \leq i \leq m) \tag{21}$$

with $\overline{a}^i = Q^{-1}a^i$.

Going through the proofs in [14], it can be verified that they hold when $\langle\,,\,\rangle_Q$ substitutes for $\langle\,,\,\rangle$, with the obvious changes. Also, [14] considers a fixed relaxation parameter α, rather than α_k, changing with the iteration. On the other hand it allows $\alpha \in (0, 2]$. Overrelaxation (i.e., $\alpha > 1$) does not work for nonquadratic objectives.

The main result of [14] is the following proposition.

PROPOSITION 3. *There exists k_0 such that for $k > k_0$,*

$$\left\|\overline{x}^{k+r} - \overline{x}^*\right\| \leq \hat{\rho}\left\|\overline{x}^k - \overline{x}^*\right\|, \tag{22}$$

where $\{\overline{x}^k\}$ is the primal sequence obtained by applying (7)–(12) to (18)–(19), $\overline{x}^*$ is the solution of (18)–(19), and $\hat{\rho}$ is a real number in $(0, 1)$ such that

$$\frac{1}{\hat{\rho}^2} = 1 + \frac{(2-\alpha)\alpha\theta^2}{1 + \alpha^2(r-1)} \tag{23}$$

and θ is defined in the following way: Let

$$\overline{I} = \{i : \langle a^i, \overline{x}^*\rangle = b_i\}, \qquad L_i = \{x : \langle a^i, x\rangle = b_i\}, \qquad V = \bigcap_{i \in \overline{I}} L_i,$$

and let L_x be the hyperplane lying farthest away from x among L_i ($i \in \overline{I}$). Then

$$\theta = \inf_{x \notin V} \frac{d(x, L_x)}{d(x, V)}, \tag{24}$$

where the distances in (24) *are those derived from the Euclidean norm.*

Proof. The proof is in Theorem 1 of [14]. □

A revision of the proof of Theorem 1 in [14] shows that when we have $\alpha_k \in (c, 1]$ instead of $\alpha \in (0, 2)$, (23) can be changed to

$$\frac{1}{\rho} = 1 + \frac{c\bar{\theta}^2}{r}, \tag{25}$$

i.e.,

$$\rho = \frac{r}{r + c\bar{\theta}^2}, \tag{26}$$

where $\bar{\theta}$ is defined as in (24) but with the distances taken with respect to $\langle\ ,\ \rangle_Q$ and $\bar{a}^i = Q^{-1}a^i$ substituting for a^i in the definition of L_i. So Proposition 3 becomes Proposition 3′.

PROPOSITION 3′. *If* $\{\bar{x}^k\}$ *is the primal sequence generated by Bregman's algorithm when applied to* (20)–(21) *and* $\bar{x}^*$ *is the solution of* (20)–(21), *then there exists* k_0 *such that for* $k > k_0$:

$$\left\|\bar{x}^{k+r} - \bar{x}^*\right\|_Q \leq \rho \left\|\bar{x}^k - \bar{x}^*\right\|_Q, \tag{27}$$

where ρ *is given by* (26).

We can find more explicit expressions or lower bounds for $\bar{\theta}$ (along the lines of [18], for instance) but we remark that all of them will naturally include Q. When we apply Proposition 3′ to Bregman's method for a general Bregman function f (§6), Q will be $\nabla^2 f(x^*)$, where x^* is the solution of (4)–(6). Since, of course, we do not know x^*, our asymptotic error constant (which can be taken as any $\bar{\rho} > \rho$, as we will prove below) cannot be estimated a priori, unless we have upper and lower bounds of the eigenvalues of $\nabla^2 f(x)$ for x in the feasible set of (4)–(6), which is rather unlikely. In practical terms, (27) means that there is a $\rho \in (0, 1)$ such that (27) holds, but does not allow for estimation of (27) until problem (4)–(6) has been solved.

5. Outline and preliminaries of the convergence rate proof. Our aim is a result similar to (27), but for the sequence resulting from application of Bregman's method to a general Bregman function f. We will proceed as follows. First we will generate the sequence $\{x^k\}$ with (7)–(12) until x^k is close enough to the solution x^* that we may approximate f by a quadratic around x^*. Starting from z^k we will consider a major iteration of the algorithm with f, producing x^{k+r} and also a major iteration with the quadratic approximation $\tilde{f}$ producing $\bar{x}^{k+r}$. For $\bar{x}^{k+r}$ we should have a relation like (27), and then we show that for x^k close enough to x^* the distance between x^{k+r} and $\bar{x}^{k+r}$ is $o(\|x^k - x^*\|)$, so that a formula like (27) holds for some $\rho' \in (\rho, 1)$, with x^{k+r} and x^k instead of $\bar{x}^{k+r}$ and $\bar{x}^k$.

In order to proceed along this path, two technical issues must be dealt with. First, for a quadratic approximation of f to hold at the solution x^*, we need differentiability at x^*. So we impose an additional hypothesis on problem (4)–(6):

(vii) f is twice continuously differentiable at the solution x^* of (4)–(6), and $\nabla^2 f(x^*)$ is nonsingular.

This condition is rather harmless if $x^* \in S$: most Bregman functions of interest are analytical in the interior of their domains. But x^* may be in the boundary of S, where f is required to be just continuous. In fact, technical hypotheses (iii), (iv), (v)

are required only for the case of x^* in the boundary and f nondifferentiable there, i.e., $D_f(x,y)$ undefined for $y \in \overline{S} - S$. We will further elaborate upon the consequences of (vii) in §8.

Second, Proposition 3′ states that (27) holds only for k larger than a certain k_0. If we proceed as announced, i.e., switching from f to $\tilde{f}$ at a certain iteration after using f up until then, we are indeed starting with the algorithm for $\tilde{f}$ at that point, and we cannot use Proposition 3′ right away. So we must once again restate Proposition 3 in more appropriate terms.

A careful perusal of the proof of Theorem 1 of [14] shows that, in fact, (27) holds for any k which satisfies conditions (i), (ii) and (iii) below (Lemmas 1 and 2 of [14] prove that for the quadratic case, there exists k_0 such that (i), (ii), and (iii) are satisfied for $k > k_0$; for the nonquadratic case we will present a new proof).

Let $\overline{x}^*$ be the solution of (20)–(21) and $I = \{i : \langle a^i, \overline{x}^* \rangle < b_i\}$. Let $\overline{T} = \{z \geq 0 : A^T z = -Q(\overline{x}^* - \hat{x})\}$, $\hat{I}_k = \{i : \overline{z}_i^{k+r} = 0\}$.

(i) $\overline{z}_i^k = 0$ for $i \in I$.

(ii) $\langle a^i, \overline{x}^{k+s} \rangle < b_i$ for $i \in I$, $0 \leq s \leq r$.

(iii) There exists $\overline{z} \in \overline{T}$ such that $\overline{z}_i = 0$ for all $i \in \hat{I}_k$, where $\{\overline{x}^k\}$, $\{\overline{z}^k\}$ is the sequence generated by Bregman's algorithm for problem (20)–(21).

We state that the proof of Theorem 1 in [14] makes it possible to rephrase the theorem as Proposition 3″.

PROPOSITION 3″. *If k is such that* (i), (ii), *and* (iii) *hold, then*

$$\left\|\overline{x}^{k+r} - \overline{x}^*\right\|_Q \leq \rho \left\|\overline{x}^k - \overline{x}^*\right\|_Q, \tag{28}$$

with ρ as in (26).

The next two propositions on the sequence $\{x^k\}$ for a general f will be used to show that when we switch from f to the quadratic $\tilde{f}$ at iteration k, x^k and z^k will be such that we can apply Proposition 3″ to the algorithm for $\tilde{f}$ starting with z^k.

Let $\{x^k\}$, $\{z^k\}$ be the sequences generated by (7)–(12) for a Bregman function f. Let x^* be the solution of problem (4)–(6), $I = \{i : \langle a^i, x^* \rangle < b_i\}$.

PROPOSITION 4. *There exists $\overline{k}_0$ such that for $k > \overline{k}_0$,*

$$z_i^k = 0 \quad \text{for} \quad i \in I. \tag{29}$$

Proof. Since $\lim_{k\to\infty} x^k = x^*$, by Proposition 1(iv),

$$\lim_{k\to\infty} (b_i - \langle a^i, x^k \rangle) = b_i - \langle a^i, x^* \rangle \tag{30}$$

$$\lim_{k\to\infty} \|x^k - x^{k+1}\| = 0. \tag{31}$$

Use (30), (31) to find $\overline{k}_0$ such that, for $k > \overline{k}_0 - r$ and $i \in I$:

$$b_i - \langle a^i, x^k \rangle > \frac{1}{2}(b_i - \langle a^i, x^* \rangle) \tag{32}$$

$$\|x^k - x^{k+1}\| < \frac{c}{3\|a^i\|}(b_i - \langle a^i, x^* \rangle). \tag{33}$$

Note that the ith component of z^k changes only at iterations k such that $i(k) = i$ and that if $\gamma_k \neq \mu_k$, then $z_{i(k)}^{k+1} = 0$, because of (9), (12). Take $i \in I$ and $k > \bar{k}_0$. Let ℓ be the last iteration before k such that $i(\ell) = i$. If $\gamma_\ell \neq \mu_\ell$, then $z_i^{\ell+1} = 0$, which implies $z_i^k = 0$, as required.

Assume $\gamma_\ell = \mu_\ell$. Then (7) and (10) are identical and by uniqueness of the Bregman projection, $x^{\ell+1}$ satisfies (7) and (8); so

$$\begin{aligned} &\langle x^{\ell+1} - x^\ell, a^i \rangle = \alpha_\ell (b_i - \langle a^i, x^\ell \rangle) > c(b_i - \langle a^i, x^\ell \rangle) \Rightarrow \\ &\|x^{\ell+1} - x^\ell\| \geq \frac{c}{\|a^i\|}(b_i - \langle a^i, x^\ell \rangle). \end{aligned} \tag{34}$$

By almost cyclicality, $\ell > \bar{k}_0 - r$, so we may use ℓ in (32), (33) which, combined with (34), produces

$$\frac{c}{3\|a^i\|}(b_i - \langle a^i, x^* \rangle) \geq \frac{c}{2\|a^i\|}(b_i - \langle a^i, x^* \rangle) > 0,$$

a contradiction. So $\gamma_\ell \neq \mu_\ell$ and the result holds. □

Let x^* be the solution of (4)–(6) and $T = \{z : \nabla f(x^*) = -A^T z\}$. Let $M = \{1, \cdots, m\}$. For $J \subset M$ let $V_J = \{z : z_i = 0 \text{ for all } i \in J\}$, $\tau_J = d(T, V_J) = \inf\{\|z - z'\| : z \in T, z' \in V_J\}$. Let $\tau = \min\{\tau_J : J \subset M, \tau_J > 0\}$. If $\tau_J = 0$ for all $J \subset M$ set $\tau = \infty$. Let $\{x^k\}$, $\{z^k\}$ be the primal and dual sequences generated by Bregman's algorithm for problem (4)–(6). Define $\bar{I}_k = \{i : z_i^{k+r} \leq \tau/2m\}$.

PROPOSITION 5. *There exists $\tilde{k}_0$ such that, for $k > \tilde{k}_0$, there exists $z \in T$ with $z_i = 0$ for all $i \in \bar{I}_k$.*

Proof. Two cases are immediate. If $\bar{I}_k = \phi$ for large enough k, any $z \in T$ works and the result holds. If $\tau = \infty$, $\bar{I}_k = M$ for all k, i.e., we need $0 \in T$. Note that $\tau = \infty$ implies $d(T, V_J) = 0$ for $J = M$, in which case $V_J = \{0\}$, $d(T, \{0\}) = 0$. Since T is a polyhedron, it follows that $0 \in T$ and the result holds with $\tilde{k}_0 = 0$.

Next, assume the result is false. This means that there exists a subsequence with indices $\{j_k\}$ such that for all $z \in T$, $z_i \neq 0$ for some $i \in \bar{I}_{j_k}$. Since the set of possible $\bar{I}_{j_k}$'s is finite, one of them occurs infinitely often and we may assume without loss of generality that $\bar{I}_{j_k} = \bar{I}_{j_{k+1}} = \cdots = J \neq \phi$. So, no $z \in T$ has $z_i = 0$ for all $i \in J$, i.e., $T \cap V_J = \phi$. Since T and V_J are polyhedra, it follows that $d(T, V_J) \neq 0$, implying $d(T, V_J) \leq \tau$.

Define $\hat{z}^k \in V_J$ as:

$$\hat{z}_i^k = \begin{cases} 0 & \text{if } i \in J \\ z_i^{j_k+r} & \text{if } i \notin J. \end{cases}$$

Since $J = \bar{I}_{j_k}$, $z_i^{j_k+r} < \tau/2m$ for $i \in J$. Then,

$$\|\hat{z}^k - z^{j_k+r}\| \leq \frac{|J|\tau}{2m} \leq \frac{\tau}{2}. \tag{35}$$

Take any $z \in T$ and use (35) to obtain

$$\tau \leq d(T, V_J) \leq \|\hat{z}^k - z\| \leq \|\hat{z}^k - z^{j_k+r}\| + \|z^{j_k+r} - z\| \leq \frac{\tau}{2} + \|z^{j_k+r} - z\|,$$

implying $\|z^{j_k+r} - z\| \geq \tau/2$. Since z is any element of T, conclude that

$$d(z^{j_k+r}, T) \geq \frac{\tau}{2}. \tag{36}$$

From Proposition 1(iv), $\lim_{k\to\infty} x^k = x^*$, from Proposition 1(i), $\nabla f(x^k) = -A^T z^k$. It follows that $\lim_{k\to\infty} A^T z^k = -\nabla f(x^*)$, which implies $\lim_{k\to\infty} d(z^k, T) = 0$ and henceforth $\lim_{k\to\infty} d(z^{j_k+r}, T) = 0$, in contradiction with (36). □

Although these last two propositions are rather technical, they become more meaningful if we think that

$$z^k \underset{k\to\infty}{\longrightarrow} z^*,$$

a vector of Kuhn–Tucker multipliers for problem (4)–(6), which will be proved in §7. Clearly, $z_i^* = 0$ for $i \in I$ by complementarity, so that

$$z_i^k \underset{k\to\infty}{\longrightarrow} 0 \quad \text{for } i \in I \quad \text{and if } z_i^{j_k+r} \underset{k\to\infty}{\longrightarrow} 0, \quad \text{then } z_i^* = 0.$$

The linear nature of the constraints allows for the stronger statements of Propositions 4 and 5: For $i \in I$, z_i^k reaches 0 at a finite k and there is a bound (namely, $\tau/2m$) such that if z_i^k is infinitely often below it, then $z_i^* = 0$. The formulation and proof of both propositions becomes somewhat cumbersome because we do not yet know that $\{z^k\}$ converges.

6. Proof of linearity of the convergence rate. First, we present a lemma on the closeness of the quadratic approximation of f. Informally, the lemma works as follows: we are going to make an iteration of (7)–(12) with respect to a hyperplane. We take a point u^* in the hyperplane and call $\tilde{f}$ the quadratic approximation of f around u^*. Consider two pairs (x^ℓ, z^ℓ) $(\ell = 1, 2)$ which will be used in an iteration of (7)–(12), with the x's as primals and the z's as duals. We iterate with (x^1, z^1) using f and with (x^2, z^2) using $\tilde{f}$. The lemma states that if x^2 is close enough to u^* and the distance between both pairs is $o(\|x^2 - u^*\|)$, then the distance between the two pairs resulting from the iteration will still be $o(\|x^2 - u^*\|)$. The nature of (7)–(12) makes this rather intuitive, but the formal statement is somewhat involved. The proof is even more so, and we relegate it to an appendix.

We start by presenting an iteration of the algorithm in "operator" form. Let f be a Bregman function with zone S, $L = \{x : \langle a, x\rangle = \beta\}$ a hyperplane such that f is strongly zone consistent with respect to it ($a \in R^n, \beta \in R$), $i \in M$, and $\alpha \in (0, 1]$. Define $P_f : S \times R^m_{\geq 0} \to S \times R^m_{\geq 0}$ in the following way:

(i) Take $(u, v) \in S \times R^m_{\geq 0}$.

(ii) Solve in t, μ:

$$\nabla f(t) - \mu a = \nabla f(u) \tag{37}$$

$$\langle a, t\rangle = \alpha\beta + (1 - \alpha)\langle a, u\rangle . \tag{38}$$

(iii) Take $\gamma = \min\{\mu, v_i\}$.

(iv) Solve in y: $\nabla f(y) = \nabla f(u) + \gamma a$.

(v) Take $w = v - \gamma e^i$. Then $(y, w) = P_f(u, v)$.

As observed after equation (2) of this paper Lemma 2.2 of [7] together with the stong zone consistence hypothesis guarantee that P_f is well defined.

Take $u^* \in L \cap \overline{S}$ such that f is twice continuously differentiable at u^*. Define

$$\tilde{f}(u) = f(u^*) + \nabla f(u^*)^T(u - u^*) + \frac{1}{2}(u - u^*)^T \nabla^2 f(u^*)(u - u^*). \tag{39}$$

Call $H = \nabla^2 f(u^*)$. H is symmetric and positive-definite by Definition 1(ii) and hypothesis (vii). For a matrix B, let

$$\|B\|_H = \sup\left(\frac{\|Bx\|_H}{\|x\|_H}\right),$$

where $\| \ \|_H$ is as defined in §3. Note that $\|H^{-1}\|_H = \|H^{-1}\|$. Let $\bar{a} = H^{-1}a$, where a is the vector which defines L.

LEMMA 2. *With the notation above, take* $(u^\ell, v^\ell) \in R^n \times R^m_{\geq 0}$ $(\ell = 1, 2)$; $\bar{\eta}, \sigma$, $\hat{\varepsilon}, \bar{\varepsilon} \in R_{>0}$, *and a neighborhood* U *of* u^* *such that:*

(i)
$$\left\|\nabla \tilde{f}(u) - \nabla f(u)\right\|_H \leq \frac{\hat{\varepsilon}}{\|H^{-1}\|} \|u - u^*\|_H \quad \text{for all } u \in U,$$

(ii) $u^1 \in U \cap S$,
(iii) $\|u^2 - u^*\|_H \leq \sigma$,
(iv) $\|u^1 - u^2\|_H \leq \bar{\varepsilon}\sigma$,
(v) $\|v^1 - v^2\| \leq \bar{\varepsilon}\sigma$,
(vi) $\hat{\varepsilon} < \frac{1}{10}$.

Let $(y^1, w^1) = P_f(u^1, v^1)$, $(y^2, w^2) = P_{\tilde{f}}(u^2, v^2)$, *with* $\tilde{f}$ *as in* (39), *and assume:*

$$\|\bar{a}\|_H = 1. \tag{40}$$

Then:

$$\|y^1 - y^2\|_H \leq 6(\hat{\varepsilon} + \bar{\varepsilon})\sigma, \qquad \|w^1 - w^2\| \leq \frac{6(\hat{\varepsilon} + \bar{\varepsilon})\sigma}{\bar{\eta}}. \tag{41}$$

This is proved in the Appendix. Note that the existence of U is guaranteed by the hypothesis of smoothness of f at u^*. □

Next we prove that the solution x^* of (4)–(6) is not altered if we substitute f by its approximation $\tilde{f}$ as in (39) at x^*.

PROPOSITION 6. *Let* x^* *be the solution of* (4)–(6) *and take* $\tilde{f}$ *as in* (39) *with* x^* *instead of* u^*. *Then* x^* *is the solution of* $\min \tilde{f}(x)$ *such that* $Ax \leq b$.

Proof. The proof is elementary. □

Now we prove that the rate of convergence of Bregman's algorithm is linear.

Consider problem (4)–(6). Assume f, A, b satisfy hypotheses (i)–(vii). Let (x^k, z^k) be the primal and dual vectors generated by (7)–(12), and x^* be the solution of (4)–(6). Let ρ be the asymptotic error constant, as in (26), of algorithm (7)–(12) when applied to the quadratic function $\tilde{f}$ defined by

$$\tilde{f}(x) = f(x^*) + \nabla f(x^*)^T(x - x^*) + \tfrac{1}{2}(x - x^*)^T \nabla^2 f(x^*)(x - x^*). \tag{42}$$

Take any $\bar{\rho}$ such that:

$$\rho < \bar{\rho} < 1. \tag{43}$$

Define

$$\lambda_s = \frac{3 \times 6^{s+1} - 2 \times 6^s - 6}{5} \qquad (0 \leq s). \tag{44}$$

Note that

$$\lambda_0 = 2, \tag{45}$$

$$\lambda_{s+1} = 6(\lambda_s + 1). \tag{46}$$

Define

$$\varepsilon = \frac{1}{2\lambda_r}(\overline{\rho} - \rho), \tag{47}$$

where r is the constant of almost cyclicality.

Let $H = \nabla^2 f(x^*)$. Define

$$\overline{a}^i = H^{-1}a^i \qquad (1 \leq i \leq m). \tag{48}$$

Observe that the primal sequence generated by Bregman's algorithm is invariant through scaling of A, b. More precisely, if $\{x^k\}, \{z^k\}$ are sequences generated by (7)–(12) for problem (4)–(6), then $\{x^k\}$, $\{\hat{z}^k\}$ will be the sequences generated by (7)–(12), starting at $\hat{z}^0$, for the problem $\min f(x)$ such that $\hat{A}x \leq \hat{b}$, $x \in \overline{S}$, where $\hat{A} = \eta^T A$, $\hat{b}_i = \eta_i b_i (1 \leq i \leq m)$, $\hat{z}_i^k = \eta_i z_i^k$ $(1 \leq i \leq m, k \geq 0)$, and $\eta \in R^m, \eta > 0$. Thus, for the study of the convergence rate of the primal sequence we may assume, without loss of generality, that:

$$\left\|\overline{a}^i\right\|_H = 1 \qquad (1 \leq i \leq m). \tag{49}$$

Also, $\{z^k\}$ converges if and only if $\{\hat{z}^k\}$ converges, so (49) entails no loss of generality regarding convergence of the dual sequence.

For $\delta > 0$, let $U_\delta = \{x : \|x - x^*\|_H \leq \delta\}$. Let $I = \{i \in M : \langle a^i, x^* \rangle < b_i\}$. Take $\delta > 0$ such that:

(i) For $\tilde{f}$ as in (42), ε as in (47), and all $x \in U_\delta$,

$$\left\|\nabla \tilde{f}(x) - \nabla f(x)\right\|_H \leq \frac{\varepsilon \|x - x^*\|_H}{\|H^{-1}\|}. \tag{50}$$

(ii) For all $i \in I$, $x \in U_{\delta(1+\varepsilon)}$,

$$\langle a^i, x \rangle < b_i, \tag{51}$$

where ε is as in (47).

(iii)

$$\delta < \frac{\tau}{2m} \tag{52}$$

with τ as defined before Proposition 4. Take k_0 such that

(iv)

$$k_0 > \max\{\overline{k}_0, \tilde{k}_0\} \tag{53}$$

with $\overline{k}_0, \tilde{k}_0$ as in Propositions 4 and 5.

(v) For $k \geq k_0$,

$$x^k \in U_\delta. \tag{54}$$

The existence of δ satisfying (50) results from hypothesis (vii), and the existence of k_0 satisfying (54) results from Proposition 1(iv).

THEOREM 1. *For $k \geq k_0$ defined as above and $\bar{\rho}$ as in* (43),

$$\|x^{k+r} - x^*\|_H \leq \bar{\rho}\,\|x^k - x^*\|_H .$$

Proof. Take $k \geq k_0$. We will consider two pair of sequences: (x^{k+s}, z^{k+s}), $(\bar{x}^{k+s}, \bar{z}^{k+s})$ $(0 \leq s \leq r)$. The first one is already defined and consists of applying (7)–(12) to f. The second one applies (7)–(12) to $\tilde{f}$ as in (42), starting at

$$\bar{z}^k = z^k \tag{55}$$

$$\bar{x}^k = x^* - H^{-1}(\nabla f(x^*) + A^T z^k). \tag{56}$$

Since

$$\nabla \tilde{f}(x) = \nabla f(x^*) + H(x - x^*), \tag{57}$$

it follows that $\nabla \tilde{f}(\bar{x}^k) = -A^T \bar{z}^k$, i.e., $(\bar{x}^k, \bar{z}^k)$ satisfy the initialization condition for algorithm (7)–(12) with $\tilde{f}$.

The proof consists of three parts, which prove the following three statements:

(I)
$$\|\bar{x}^{k+s+1} - x^*\|_H \leq \|\bar{x}^{k+s} - x^*\|_H \qquad (0 \leq s \leq r-1),$$
$$\bar{x}^{k+s} \in U_{\delta(1+\varepsilon)} \qquad (0 \leq s \leq r).$$

(II)
$$\|x^{k+r} - \bar{x}^{k+r}\|_H \leq \frac{2}{3}(\bar{\rho} - \rho)\,\|x^k - x^*\|_H .$$

(III)
$$\|\bar{x}^{k+r} - x^*\|_H \leq \frac{1}{3}(\bar{\rho} + 2\rho)\,\|x^k - x^*\|_H .$$

Before proceeding to the proof of (I), (II), (III), we note that (II) and (III) imply the result of the theorem:

$$\|x^{k+r} - x^*\|_H \leq \|x^{k+r} - \bar{x}^{k+r}\|_H + \|\bar{x}^{k+r} - x^*\|_H \leq \bar{\rho}\,\|x^k - x^*\|_H .$$

Proof of (I). First we show that $\bar{x}^k \in U_{\delta(1+\varepsilon)}$. As observed before, $\nabla \tilde{f}(\bar{x}^k) = -A^T \bar{z}^k$. From (55) and Proposition 1(i), conclude that $\nabla \tilde{f}(\bar{x}^k) = \nabla f(x^k)$. So,

$$\begin{aligned} H(x^k - \bar{x}^k) = \nabla \tilde{f}(x^k) - \nabla \tilde{f}(\bar{x}^k) = \nabla \tilde{f}(x^k) - \nabla f(x^k) \Rightarrow \\ x^k - \bar{x}^k = H^{-1}(\nabla \tilde{f}(x^k) - \nabla f(x^k)). \end{aligned} \tag{58}$$

Using (50) and (54) in (58),

$$\begin{aligned} &\|x^k - \bar{x}^k\|_H \leq \varepsilon\,\|x^k - x^*\|_H \\ &\|\bar{x}^k - x^*\|_H \leq \|\bar{x}^k - x^k\|_H + \|x^k - x^*\|_H \leq (1+\varepsilon)\,\|x^k - x^*\|_H \leq (1+\varepsilon)\delta, \end{aligned} \tag{59}$$

i.e., $\bar{x}_k \in U_{\delta(1+\varepsilon)}$.

From (51),

$$\langle a^i, \bar{x}^k \rangle < b_i \quad \text{for } i \in I. \tag{60}$$

Next we consider two cases:

(i) If $i(k) \in I$, from (53), (55), and Proposition 4,

$$\overline{z}^k_{i(k)} = z^k_{i(k)} = 0. \tag{61}$$

Take $\overline{\mu}_k$ and $\overline{\gamma}_k$ as the values of μ_k, γ_k obtained in (7), (9) when (7)–(12) is applied to $\overline{x}^k$ with f. From (6) and Proposition 1(v), $\overline{\mu}_k \geq 0$. From (9) and (61), $\overline{\gamma}_k = 0$ and from (10), (12),

$$\overline{z}^{k+1} = \overline{z}^k, \qquad \overline{x}^{k+1} = \overline{x}^k.$$

So $\left\|\overline{x}^{k+1} - x^*\right\|_H = \left\|\overline{x}^k - x^*\right\|_H$ and $\overline{x}^{k+1} \in U_{\delta(1+\varepsilon)}$.

(ii) If $i(k) \notin I$, then $x^* \in L_{i(k)}$ and from Lemma 1(i),

$$\left\|\overline{x}^{k+1} - x^*\right\|_H \leq \left\|\overline{x}^k - x^*\right\|_H$$

and

$$\overline{x}^{k+1} \in U_{\delta(1+\varepsilon)}, \qquad \overline{z}^{k+1}_i = 0 \quad \text{for } i \in I.$$

In both cases, we get $\left\|\overline{x}^{k+1} - x^*\right\|_H \leq \left\|\overline{x}^k - x^*\right\|_H$, $\overline{x}^{k+1} \in U_{\delta(1+\varepsilon)}$, $\overline{z}^{k+1}_i = 0$ for $i \in I$.

So we may use recurrently the arguments of (i), (ii) for x^{k+s} $(1 \leq s \leq r-1)$ and the result follows.

Proof of (II). Define

$$\varepsilon_s = \lambda_s \varepsilon, \tag{62}$$

with λ_s, ε as in (44), (47). We will use Lemma 2 inductively to prove the following statements:

(i)

$$\left\|x^{k+s} - \overline{x}^{k+s}\right\|_H \leq \varepsilon_s \left\|\overline{x}^k - x^*\right\|_H \qquad (0 \leq s \leq r), \tag{63}$$

(ii)

$$\left\|z^{k+s} - \overline{z}^{k+s}\right\| \leq \varepsilon_s \left\|\overline{x}^k - x^*\right\|_H \qquad (0 \leq s \leq r). \tag{64}$$

(a) $s = 0$: (ii) is trivial because $z^k = \overline{z}^k$. For i, from (59),

$$\left\|\overline{x}^k - x^k\right\|_H \leq \varepsilon \left\|x^k - x^*\right\|_H \leq \varepsilon(\left\|x^k - \overline{x}^k\right\|_H + \left\|\overline{x}^k - x^*\right\|_H) \Rightarrow$$
$$\left\|\overline{x}^k - x^k\right\|_H \leq \frac{\varepsilon}{1-\varepsilon} \left\|\overline{x}^k - x^*\right\|_H \leq 2\varepsilon \left\|\overline{x}^k - x^*\right\|_H = \varepsilon_0 \left\|\overline{x}^k - x^*\right\|_H.$$

(b) Assume true for $s \leq r-1$.

Consider two cases. If $i(k+s) \in I$, then $\overline{x}^{k+s+1} = \overline{x}^{k+s}$, $\overline{z}^{k+s+1} = \overline{z}^{k+s}$ as in case (i) of part (I). The same argument holds for the nonquadratic sequences $\{x^k\}$, $\{z^k\}$: by Proposition 4, $\overline{z}^{k+s}_{i(k)} = 0$; by (54) and (51), $\left\langle a^{i(k+s)}, x^{k+s} \right\rangle < b_{i(k+s)}$, so that $\mu_{k+s} \geq 0$, implying $\gamma_{k+s} = 0$ and therefore $x^{k+s+1} = x^{k+s}$, $z^{k+s+1} = z^{k+s}$. So, using the inductive hypothesis,

$$\left\|\overline{x}^{k+s+1} - x^{k+s+1}\right\|_H = \left\|\overline{x}^{k+s} - x^{k+s}\right\|_H \leq \varepsilon_s \left\|\overline{x}^k - x^*\right\|_H \leq \varepsilon_{s+1} \left\|\overline{x}^k - x^*\right\|,$$

observing that $\varepsilon_s < \varepsilon_{s+1}$ by (46). The same argument applies for (64). □

If $i(k+s) \notin I$, we will apply Lemma 2 with

$$u^* = x^*, \quad U = U_\delta, \quad \sigma = \left\|\overline{x}^k - x^*\right\|_H,$$
$$\hat{\varepsilon} = \varepsilon, \quad \overline{\varepsilon} = \varepsilon_s, \quad i = i(k+s), \quad a = a^{i(k+s)},$$
$$\beta = b_{i(k+s)}, \quad \alpha = \alpha_{k+s}, \quad L = L_{i(k)}, \quad u^1 = x^{k+s},$$
$$u^2 = \overline{x}^{k+s}, \quad v^1 = z^{k+s}, \quad v^2 = \overline{z}^{k+s}.$$

We must check the hypotheses of Lemma 2: $x^* \in L_{i(k+s)}$ because $i(k+s) \notin I$, (i) holds because of (50), (ii) because of (54) and strong zone consistence, (iii) means $\left\|\overline{x}^{k+s} - x^*\right\|_H \leq \left\|\overline{x}^k - x^*\right\|_H$, which is true because of part (I), (iv) and (v) are the inductive hypotheses, and (vi) follows from (44)–(47). (40) follows from (49). Next, observe that the definitions of P_f, $P_{\tilde{f}}$ imply that

$$y^1 = x^{k+s+1}, \qquad y^2 = \overline{x}^{k+s+1}, \qquad w^1 = z^{k+s+1}, \qquad w^2 = \overline{z}^{k+s+1}.$$

Conclude from Lemma 2 and (46), (62) that

$$\begin{aligned}\left\|x^{k+s+1} - \overline{x}^{k+s+1}\right\|_H &\leq 6(\varepsilon + \varepsilon_s)\left\|\overline{x}^k - x^*\right\|_H \\ &= 6(\lambda_s + 1)\varepsilon\left\|\overline{x}^k - x^*\right\|_H = \varepsilon_{s+1}\left\|\overline{x}^k - x^*\right\|_H.\end{aligned}$$

The same argument applies for (64).

(63) and (64) are therefore established and we look at (63) with $s = r$:

$$\left\|x^{k+r} - \overline{x}^{k+r}\right\|_H \leq \varepsilon_r\left\|\overline{x}^k - x^*\right\|_H = \lambda_r\varepsilon\left\|\overline{x}^k - x^*\right\|_H. \tag{65}$$

Using (59) in (65),

(66)

$$\left\|x^{k+r} - \overline{x}^{k+r}\right\|_H \leq \lambda_r\varepsilon(1+\varepsilon)\left\|x^k - x^*\right\|_H \leq \frac{4}{3}\lambda_r\varepsilon\left\|x^k - x^*\right\|_H = \frac{2}{3}(\overline{\rho}-\rho)\left\|x^k - x^*\right\|_H,$$

because (43)–(47) imply $\varepsilon < \frac{1}{3}$. (II) is proved.

Proof of (III). We will apply Proposition 3″ to the sequences $\{\overline{x}^k\}$, $\{\overline{z}^k\}$. As noted before, $(\overline{x}^k, \overline{z}^k)$ is an admissible starting pair for application of (7)–(12) to $\tilde{f}$. We need to check that conditions (i), (ii) and (iii) of §5 are satisfied. (i) follows from (53), (55) and Proposition 4.

(ii) follows from Part (I) and (51). We prove (iii): Take $i \in \hat{I}_k$, so that $\overline{z}_i^{k+r} = 0$. Then using (64), with $s = r$, and the argument of (66),

$$\begin{aligned}z_i^{k+r} = z_i^{k+r} - \overline{z}_i^{k+r} &\leq \left\|z^{k+r} - \overline{z}^{k+r}\right\| \leq \varepsilon_r\left\|\overline{x}^k - x^*\right\|_H \\ &\leq \frac{2}{3}(\overline{\rho} - \rho)\left\|x^k - x^*\right\|_H \leq \left\|x^k - x^*\right\|_H \leq \delta \leq \frac{\tau}{2m}.\end{aligned} \tag{67}$$

The last two inequalities of (67) result from (54) and (52).

From (67), $i \in \overline{I}_k$. We have shown that $\hat{I}_k \subset \overline{I}_k$. Now use (53) and Proposition 5 to conclude that there exists $z \in T$ such that $z_i = 0$ for all $i \in \overline{I}_k$, and therefore $z_i = 0$ for all $i \in \hat{I}_k$. We claim that such z can be used in (iii) because $T = \overline{T}$: Remember that $T = \{z: A^T z = -\nabla f(x^*)\}$, $\overline{T} = \{z: A^T z = \nabla\tilde{f}(\overline{x}^*)\}$ where x^* and $\overline{x}^*$ are the solutions of the respective problems. By Proposition 6, $x^* = \overline{x}^*$, and from

(57), $\nabla f(x^*) = \nabla \tilde{f}(x^*)$. So the conclusion of Proposition 3'' holds and, again using Proposition 6, (59), $\rho < 1$ from (43), and $\lambda_r \geq \frac{3}{2}$ from (45),

$$
\begin{aligned}
\left\|\overline{x}^{k+r} - x^*\right\|_H &\leq \rho \left\|\overline{x}^k - x^*\right\|_H \leq \rho(1+\varepsilon) \left\|x^k - x^*\right\|_H \\
&\leq (\rho+\varepsilon) \left\|x^k - x^*\right\|_H = \left[\rho + \frac{1}{2\lambda_r}(\overline{\rho} - \rho)\right] \left\|x^k - x^*\right\|_H \\
&\leq \left[\rho + \frac{1}{3}(\overline{\rho} - \rho)\right] \left\|x^k - x^*\right\|_H = \frac{1}{3}(\overline{\rho} + 2\rho) \left\|x^k - x^*\right\|_H .
\end{aligned} \tag{68}
$$

Part (III) has been proved. □

The use of $\| \ \|_H$ in the theorem is just a matter of convenience. If ζ, ζ' are the smallest and the largest eigenvalues of H, we have $\zeta \|x\| \leq \|x\|_H \leq \zeta' \|x\|$; therefore, iterating the inequality of the theorem, for any k,

$$
\zeta \left\|x^{k_0+kr} - x^*\right\| \leq \left\|x^{k_0+kr} - x^*\right\|_H \leq \overline{\rho}^k \left\|x^{k_0} - x^*\right\|_H \leq \zeta' \left\|x^{k_0} - x^*\right\| \overline{\rho}^k,
$$

which implies $\|x^{k_0+kr} - x^*\| \leq C \|x^{k_0} - x^*\| \overline{\rho}^k$ where C is the condition number of H, i.e., the asymptotic error constant is the same for the Euclidean norm: $\|x^k - x^*\|$ goes to zero as $\overline{\rho}^k$.

7. Convergence of the dual sequence. One of the main consequences of the linear rate of convergence is the convergence of the dual sequence $\{z^k\}$. We need a preliminary result.

Take ε as in (47), ε_1 as in (62). Define

$$
\nu = (1+\varepsilon_1)(1+\varepsilon). \tag{69}
$$

PROPOSITION 7. *Take k_0 satisfying* (53), (54), *x^k, γ_k defined by* (7)–(12), *and x^* the solution of* (4)–(6). *If the hypotheses of Theorem* 1 *hold, then*

$$
|\gamma_k| \leq \nu \left\|x^k - x^*\right\|_H . \tag{70}
$$

Proof. If $i(k) \in I$, it follows from (53), Proposition 4, and Proposition 1(v) that $z^k_{i(k)} = 0$, $\mu_k \geq 0$; so $\gamma_k = 0$ and (70) holds.

Assume $i(k) \notin I$. So $x^* \in L_{i(k)}$. Consider the sequence $\{\overline{z}^k\}$ as defined in Theorem 1. Use (55), (64) with $s = 1$, Lemma 1(ii), and (59), to obtain

$$
\begin{aligned}
|\gamma_k| = \left\|z^{k+1} - z^k\right\| &\leq \left\|z^{k+1} - \overline{z}^{k+1}\right\| + \left\|\overline{z}^{k+1} - \overline{z}^k\right\| = \left\|z^{k+1} - \overline{z}^{k+1}\right\| + \overline{\gamma}_k \\
&\leq \varepsilon_1 \left\|\overline{x}^k - x^*\right\|_H + \frac{1}{\left\|\overline{a}^{i(k)}\right\|_H} \left\|\overline{x}^k - x^*\right\|_H \leq \varepsilon_1 \left\|\overline{x}^k - x^*\right\|_H + \left\|\overline{x}^k - x^*\right\|_H \\
&= (1+\varepsilon_1) \left\|\overline{x}^k - x^*\right\|_H \leq (1+\varepsilon_1)(1+\varepsilon) \left\|x^k - x^*\right\|_H .
\end{aligned}
$$

□

THEOREM 2. *If problem* (4)–(6) *satisfies hypotheses* (i)–(vii), *then the dual sequence $\{z^k\}$ generated by* (7)–(12) *converges to an optimal dual solution, i.e., to a vector $z^* \in R^m \geq 0$, which satisfies*

$$
\nabla f(x^*) = -A^T z^* \tag{71}
$$

$$
(Ax^* - b)^T z^* = 0, \tag{72}
$$

where x^* is the solution of problem (4)–(6).

Proof. Take $i \in M$, k_0 as in Theorem 1, and let $K_i = \{\ell \geq 0 : i(k_0 + \ell) = i\}$.

The ith component of the dual vector is updated by substraction of γ_k at those iterations such that $i(k) = i$, so

$$z_i^{k_0+kr} = z_i^{k_0} - \sum_{\substack{\ell=0 \\ \ell \in K_i}}^{kr-1} \gamma_{k_0+\ell}. \tag{73}$$

We claim that the series in the right-hand side of (73) is absolutely convergent, hence the following is convergent:

$$\sum_{\substack{\ell=0 \\ \ell \in K_i}}^{kr-1} |\gamma_{k_0+\ell}| \leq \sum_{\ell=0}^{kr-1} |\gamma_{k_0+\ell}| = \sum_{j=0}^{k-1} \sum_{s=0}^{r-1} |\gamma_{k_0+jr+s}| \leq \nu \sum_{j=0}^{k-1} \sum_{s=0}^{r-1} \|x^{k_0+jr+s} - x^*\|_H$$

$$\leq \nu \sum_{j=0}^{r-1} \overline{\rho}^j \sum_{s=0}^{r-1} \|x^{k_0+s} - x^*\|_H \leq \frac{\nu}{1-\overline{\rho}} \sum_{s=0}^{r-1} \|x^{k_0+s} - x^*\|_H,$$

where we use Proposition 7 in the second inequality and Theorem 1 in a recurrent way in the next one. The claim is established. So there exists z_i^* such that

$$\lim_{k \to \infty} z_i^{k_0+kr} = z_i^*. \tag{74}$$

So variables z_i^k converge after major iterations from z^{k_0}. In order to see that the limit is the same for intermediate iterations, note that

$$\gamma_k a^{i(k)} = \nabla f(x^{k+1}) - \nabla f(x^k) \Longrightarrow |\gamma_k| \leq \frac{1}{\eta} [\nabla f(x^{k+1}) - \nabla f(x^k)] \underset{k \to \infty}{\longrightarrow} 0 \tag{75}$$

because of Proposition 1(iv) and hypothesis (vii). Therefore, for p such that $1 \leq p \leq r-1$,

$$z^{k_0+kr+p} = z_i^{k_0+kr} + \sum_{\substack{\ell=kr \\ \ell \in K_i}}^{kr+p} \gamma_{k_0+\ell}. \tag{76}$$

The first term in the right-hand side of (75) tends to z_i^* and the second one tends to 0, because the summation contains at most $r-1$ terms, each of which tends to 0 by (74). We conclude that $z_i^* = \lim_{k \to 0} z_i^k$.

From Proposition 1(i), $\nabla f(x^k) = -A^T z^k$. Taking limits as k tends to infinity,

$$\nabla f(x^*) = -A^T z^*.$$

In view of Proposition 1(ii), since $Ax^* \leq b$, (72) just means that $z_i^* = 0$ for those i such that $\langle a^i, x^* \rangle < b_i$. From Proposition 4, for $k > \overline{k}_0$, $z_i^k = 0$ for such an i, so that $z_i^* = \lim_{k \to \infty} z_i^k = 0$. □

8. Discussion on the hypothesis of smoothness at the solution. We discuss here the implications of hypothesis (vii), smoothness at the solution, on the applicability of the algorithm. Although the method is proposed for a rather general

objective function, in practice it has been applied to just a few. In the first place we have convex quadratic objectives. Obviously, they satisfy hypothesis (vii) in all R^n. Another function that has been used in connection with Bregman's algorithm is Burg's entropy, defined as $f(x) = -\sum_{j=1}^{n} \log x_j$. The zone of f is $R^n_{>0}$ and in fact f is not continuous in the boundary of S, i.e., it does not satisfy hypothesis (ii). However, Bregman's method works. It has been shown in [5] that when f is discontinuous at a certain subset $\hat{S}$ of $\overline{S} - S$, the convergence results still hold provided that the distances from the level sets $R_1(y,h)$, $R_2(x,h)$ to $\hat{S}$ are strictly positive for all x, y, h. Several special algorithms for this f, resulting from applying specific relaxation strategies in (7)–(12), are studied in [6]. It is clear that when the constraint set $\{x: Ax \leq b\}$ intersects $R^n_{>0}$, the solution x^* is strictly positive, and f is analytical at x^*, so our results apply. Otherwise, the problem is not meaningful.

Finally, the most significant example of a Bregman function may be Shannon's entropy, defined as $f(x) = \sum_{j=1}^{n} x_j \log x_j$ for $x \in R^n_{\geq 0}$, with the convention $0 \log 0 = 0$. In this case the zone is $S = R^n_{\geq 0}$ and f satisfies (i)–(vi) (which in fact were introduced with this function in mind). Hypothesis (vii) holds if x^* is strictly positive, but it does not hold if x^* is in the boundary of the positive orthant. The relevance of Bregman's method for this function is enhanced by the existence of several methods for entropy maximization, like MART, that can be shown to be particular cases of Bregman's method under a special relaxation strategy (see [4]).

We show next an example of an application of Bregman's method to f, with solution in the boundary of the positive orthant, sublinear primal convergence, and dual divergence.

Counterexample.

$$m = n = r = 2, \qquad f(x) = \sum_{j=1}^{2} x_j \log x_j, \qquad \alpha_k = 1,$$

$$A = \begin{bmatrix} 1 & 1 \\ 0 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad x^* = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We apply MART to this case. The iteration of MART for this f is:

$$\gamma_k = \min\left\{ z^k_{i(k)}, sg(b_{i(k)}) \log \frac{b_{i(k)}}{\langle a^{i(k)}, x^k \rangle} \right\}, \tag{77}$$

$$x_j^{k+1} = x_j^k \exp(\gamma_k a_j^{i(k)}), \tag{78}$$

$$z^{k+1} = z^k - \gamma_k e^{i(k)}, \tag{79}$$

with initialization

$$z^0 \in R^m_{\geq 0}, \qquad x_j^0 = \frac{-1}{1 + \sum_{j=1}^{m} a_{ij} z_i^0} \qquad (1 \leq j \leq n).$$

See [4] for the details of the reduction of Bregman's method to (77)–(79).

We take $i(k) = \{1, 2, 1, 2, 1, 2, \cdots\}$.

It can easily be shown, from (77)–(79), that

$$x_1^{2k+2} = x_1^{2k+1} = \frac{x_1^{2k}}{1 + x_1^{2k}}, \tag{80}$$

$$x_2^{2k+1} = \frac{1}{1 + x_1^{2k}}, \tag{81}$$

$$x_2^{2k+2} = 1. \tag{82}$$

It follows from (80), (82) that, calling $x_1^0 = \omega$, we have

$$x_1^{2k} = \frac{k\omega}{1 + k\omega}, \tag{83}$$

$$x_2^{2k} = 1. \tag{84}$$

So

$$x^{2k} \underset{k\to\infty}{\longrightarrow} x^*,$$

but the rate is sublinear. $\|x^{2k+2} - x^*\| = x_1^{2k}$, and assuming $x_1^{2k+2} < \rho x_1^{2k}$, with $0 < \rho < 1$, we get from (83), after some algebra, that for all large enough k,

$$0 < \frac{1}{\omega} < \frac{1}{1-\rho} - k, \tag{85}$$

which is clearly false.

It also follows from (80)–(82) that

$$z_1^{2k+2} = z_1^0 + \sum_{j=0}^{k} \log\left[\frac{1 + (j+1)\omega}{1 + j\omega}\right], \tag{86}$$

and the series in (86) diverges (its general term is greater than $1/2j$).

This counterexample shows that hypothesis (vii) is indeed essential for our results.

We remark that the counterexample above is somewhat patological, as shown in the next Proposition: The solution is in the boundary only when the whole feasible set is contained in the boundary.

PROPOSITION 8. *Consider problem* (4)–(6) *with* $f(x) = \sum_{j=1}^n x_j \log x_j$. *If there exists* $x > 0$ *such that* $Ax \leq b$, *then* $x^* > 0$.

Proof. Assume by contradiction that for the solution x^*, $J = \{j : x_j^* = 0\} \neq \phi$, but there exists $y > 0$ such that $Ay \leq b$. Take $x(h) = x^* + h(y - x^*)$, which is feasible for $h \in [0, 1]$. Let $\varphi(h) = f(x(h))$.

$$\varphi'(h) = \sum_{j \notin J} (y_j - x_j^*)\{1 + \log[x_j^* + h(y_j - x_j^*)]\} + \sum_{j \in J} y_j[1 + \log(hy_j)],$$

$\lim_{h\to 0} \varphi'(h) = -\infty$ (because the first summation is bounded below for $h \in [0, 1]$), meaning that φ decreases near $h = 0$, so $f(x(h)) < f(x^*)$ for h close to 0, in contradiction with the optimality of x^*. □

Proposition 8 means that our results hold when a Slater condition is satisfied: existence of a strictly positive feasible solution, for in such a case, x^* is strictly positive and f is analytical at x^*. It also suggests that, if it is possible to preprocess the linear system in order to detect those components x_j that have to be zero in order for x to be feasible (and of course eliminate them from the problem), it may pay to do so, because then the rate of convergence becomes linear.

Appendix.

Proof of Lemma 2. From the definitions of $P_f, P_{\tilde{f}}$ the following relations hold:

(I)

$$\begin{aligned} \nabla f(t^1) - \mu_1 a &= \nabla f(u^1), \\ \langle a, t^1 \rangle &= \alpha\beta + (1-\alpha)\langle a, u^1 \rangle, \\ \gamma_1 &= \min\{\mu_1, v_i^1\}, \\ \nabla f(y^1) &= \nabla f(u^1) + \gamma_1 a, \\ w^1 &= v^1 - \gamma_1 e^i. \end{aligned}$$

(II)

$$\begin{aligned} \nabla \tilde{f}(t^2) - \mu_2 a &= \nabla \tilde{f}(u^2), \\ \langle a, t^2 \rangle &= \alpha\beta + (1-\alpha)\langle a, u^2 \rangle, \\ \gamma_2 &= \min\{\mu_2, v_i^2\}, \\ \nabla \tilde{f}(y^2) &= \nabla \tilde{f}(u^2) + \gamma_2 a, \\ w^2 &= v^2 - \gamma_2 e^i. \end{aligned}$$

Let $\Delta(u) = \nabla f(u) - \nabla \tilde{f}(u)$. Using (57), after some algebra, (I) and (II) can be rewritten as:

(I′)

$$\begin{aligned} t^1 - \mu_1 \bar{a} &= u^1 + H^{-1}(\Delta(u^1) - \Delta(t^1)), \\ \langle \bar{a}, t^1 \rangle_H &= \alpha\beta + (1-\alpha)\langle \bar{a}, u^1 \rangle_H, \\ \gamma_1 &= \min\{\mu_1, v_i^1\}, \\ y^1 &= u^1 + \gamma_1 \bar{a} + H^{-1}(\Delta(u^1) - \Delta(y^1)), \\ \omega^1 &= v - \gamma_1 e^i. \end{aligned}$$

(II′)

$$\begin{aligned} t^2 - \mu_2 \bar{a} &= u^2, \\ \langle \bar{a}, t^2 \rangle_H &= \alpha\beta + (1-\alpha)\langle \bar{a}, u^2 \rangle_H, \\ \gamma_2 &= \min\{\mu_2, v_i^2\}, \\ y^2 &= u^2 + \gamma_2 \bar{a}, \\ w^2 &= v^2 - \gamma_2 e^i. \end{aligned}$$

Let $u = u^1 - u^2$, $v = v^1 - v^2$, $t = t^1 - t^2$, $y = y^1 - y^2$, $w = w^1 - w^2$, $\mu = \mu_1 - \mu_2$, $\gamma = \gamma_1 - \gamma_2$.

From (I′), (II′),

$$t - \mu\bar{a} = u + H^{-1}(\Delta(u^1) - \Delta(t^1)), \tag{87}$$

$$\langle \bar{a}, t \rangle_H = (1-\alpha)\langle \bar{a}, u \rangle_H, \tag{88}$$

$$\gamma = \min\{\mu_1, v_i^1\} - \min\{\mu_2, v_i^2\}, \tag{89}$$

$$y = u + \gamma\bar{a} + H^{-1}(\Delta(u^1) - \Delta(y^1)), \tag{90}$$

$$w = v - \gamma e^i. \tag{91}$$

Let $\pi_1 = \|u^1 - u^*\|_H$, $\pi_2 = \|t^1 - u^*\|_H$, $\pi_3 = \|y^1 - u^*\|_H$.
Substituting (87) in (88),

$$\mu = \frac{-\alpha \langle \bar{a}, u \rangle_H + \langle \bar{a}, H^{-1}(\Delta(u^1) - \Delta(t^1)) \rangle_H}{\|\bar{a}\|_H^2} \Longrightarrow$$

$$|\mu| \leq \frac{\|u\|_H + \|H^{-1}\| \left(\|\Delta(u^1)\|_H + \|\Delta(t^1)\|_H\right)}{\|\bar{a}\|_H} \leq \bar{\varepsilon}\sigma + \hat{\varepsilon}\pi_1 + \hat{\varepsilon}\pi_2 \tag{92}$$

using (40) and hypotheses (i) and (iv).

Using (92) in (87), together with (i), (iv), and (40),

$$\|t\|_H \leq \mu \|\bar{a}\|_H + \|u\|_H + \|H^{-1}\| \left(\|\Delta(u^1)\|_H + \|\Delta(t^1)\|_H\right) \leq 2(\bar{\varepsilon}\sigma + \hat{\varepsilon}\pi_1 + \hat{\varepsilon}\pi_2). \tag{93}$$

From (89), (91), and (v),

$$|\gamma| \leq |\mu| + \|v\| \leq 2\bar{\varepsilon}\sigma + \hat{\varepsilon}\pi_1 + \hat{\varepsilon}\pi_2. \tag{94}$$

From (90), (94), (i), and (iv),

$$\begin{aligned} \|y\|_H &\leq \|u\|_H + |\gamma| \, \|\bar{a}\|_H + \|H^{-1}\| \left(\|\Delta(u^1)\|_H + \|\Delta(y^1)\|_H\right) \\ &\leq 3\bar{\varepsilon}\sigma + 2\hat{\varepsilon}\pi_1 + \hat{\varepsilon}\pi_2 + \hat{\varepsilon}\pi_3. \end{aligned} \tag{95}$$

Next we find bounds on π_1, π_2, π_3:

$$\pi_1 = \|u^1 - u^*\|_H \leq \|u^1 - u^2\|_H + \|u^2 - u^*\|_H \leq (1 + \bar{\varepsilon})\sigma \tag{96}$$

using (iii), (iv).

$$\pi_2 = \|t^1 - u^*\|_H \leq \|t^1 - t^2\|_H + \|t^2 - u^*\|_H \leq \|t\|_H + \|u^2 - u^*\|_H . \tag{97}$$

The last inequality follows from the argument of the proof of Lemma 1(i), noting that t^2 is the orthogonal projection of u^2 with respect to $\langle \, , \, \rangle_H$ onto a hyperplane parallel to L and lying between u^2 and L.

From (97), (iii),

$$\pi_2 \leq \|t\|_H + \sigma, \tag{98}$$

$$\pi_3 = \|y^1 - u^*\|_H \leq \|y^1 - y^2\|_H + \|y^2 - u^*\|_H \leq \|y\|_H + \|u^2 - u^*\|_H \leq \|y\|_H + \sigma, \tag{99}$$

using Lemma 1(i) and (iii).

Substituting (96), (98) into (93),

$$\|t\|_H = \frac{2\sigma}{1 - 2\hat{\varepsilon}}(\bar{\varepsilon} + 2\hat{\varepsilon} + \bar{\varepsilon}\hat{\varepsilon}). \tag{100}$$

Substituting (100) into (98),

$$\pi_2 \leq \frac{\sigma}{1 - 2\hat{\varepsilon}}(1 + \bar{2}\varepsilon + 2\hat{\varepsilon} + 2\bar{\varepsilon}\hat{\varepsilon}). \tag{101}$$

Substituting (96) and (101) into (94),

$$|\gamma| \leq \frac{2\sigma}{(1-2\hat{\varepsilon})}(\overline{\varepsilon}+\hat{\varepsilon}). \tag{102}$$

Substituting (96), (99), and (101) into (95),

$$\|y\|_H \leq \frac{\sigma}{(1-2\hat{\varepsilon})(1-\hat{\varepsilon})}(3\overline{\varepsilon}+4\hat{\varepsilon}-4\overline{\varepsilon}\hat{\varepsilon}-4\hat{\varepsilon}^2-\overline{\varepsilon}\hat{\varepsilon}) \leq \frac{\sigma(3\overline{\varepsilon}+4\hat{\varepsilon})}{(1-2\hat{\varepsilon})(1-\hat{\varepsilon})}; \tag{103}$$

using (vi) in (103),

$$\|y^1-y^2\|_H = \|y\|_H \leq 6(\hat{\varepsilon}+\overline{\varepsilon})\sigma. \tag{104}$$

From (91), (102), and (v),

$$\|w\| \leq \|v\| + |\gamma| \leq \frac{\sigma(3\overline{\varepsilon}+2\hat{\varepsilon})}{(1-2\hat{\varepsilon})}; \tag{105}$$

using (vi) in (105),

$$\|w^1-w^2\| = \|w\| \leq 6(\hat{\varepsilon}+\overline{\varepsilon})\sigma \tag{106}$$

((104) and (106) are the required inequalities). □

REFERENCES

[1] A. Ben-Tal, M. Teboulle, A. Charnes, *The role of duality in optimization problems involving entropy functionals with applications to information theory,* J. Optim. Theory Appl., 58 (1988), pp. 209–223.

[2] L. Bregman, *The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming*, USSR Comput. Math. and Math. Phys., 7 (1967), pp. 200–217.

[3] Y. Censor, *Row-action methods for huge and sparse systems and their applications*, SIAM Rev., 23 (1981), pp. 444–464.

[4] Y. Censor, A. De Pierro, T. Elfving, G. Herman, and A. Iusem, *On iterative methods for linearly constrained entropy maximization*, in Numerical Analysis and Mathematical Modelling, Vol 24, Banach Center Publications, 1989, pp. 147–165.

[5] Y. Censor, A. De Pierro, and A. Iusem, *On maximization of entropies and a generalization of Bregman's method for convex programming*, Tech. Report MIPG113, Department of Radiology, University of Pennsylvania, 1986.

[6] ———, *Optimization of Burg's entropy over linear constraints*, Appl. Numer. Math., 7 (1991), pp. 151–165.

[7] Y. Censor and A. Lent, *An iterative row-action method for interval convex programming*, J. Optim. Theory Appl., 34 (1981), pp. 321–353.

[8] J. N. Darroch and D. Ratcliff, *Generalized iterative scaling for log-linear models*, Ann. Statist., 43 (1972), pp. 1470–1480.

[9] A. De Pierro and A. Iusem, *A relaxed version of Bregman's method for convex programming*, J. Optim. Theory Appl., 5 (1986), pp. 421-440.

[10] S. Erlander, *Entropy in linear programs*, Math. Programming, 21 (1981), pp. 137–151.

[11] G. Herman, *Image Reconstruction from Projections. The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.

[12] C. Hildreth, *A quadratic programming procedure*, Naval Res. Logist. Quart., 4 (1957), pp. 79–85. Erratum, Naval Res. Logist. Quart., 4 (1957), p. 361.

[13] A. Iusem and A. De Pierro, *A simultaneous iterative method for computing projections on polyhedra*, SIAM J. Control, 25 (1986), pp. 231-243.

[14] ———, *On the convergence properties of Hildreth's quadratic programming algorithm*, Math. Programming, 47 (1990), pp. 37–51.

[15] A. IUSEM AND M. TEBOULLE, *A Primal Dual Iterative Algorithm for a Maximum Likelihood Estimation Problem*, Math. Programming, to appear.

[16] B. LAMOND AND N. F. STEWART, *Bregman's balancing method*, Transportation Res. B, 15 (1981), pp. 239–249.

[17] A. LENT AND Y. CENSOR, *Extensions of Hildreth's row-action method for quadratic programming*, SIAM J. Control, 18 (1980), pp. 444–454.

[18] J. MANDEL, *Convergence of the cyclical relaxation method for linear inequalities*, Math. Programming, 30 (1984), pp. 218–228.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 425-447, November 1991

AN AUCTION ALGORITHM FOR SHORTEST PATHS*

DIMITRI P. BERTSEKAS†

Abstract. A new and simple algorithm for finding shortest paths in a directed graph is proposed. In the single origin-single destination case, the algorithm maintains a single path starting at the origin, which is extended or contracted by a single node at each iteration. Simultaneously, at most one dual variable is adjusted at each iteration so as to either improve or maintain the value of a dual function. For the case of multiple origins, the algorithm is well suited for parallel computation. It maintains multiple paths that can be extended or contracted in parallel by several processors that share the results of their computations. Based on experiments with randomly generated problems on a serial machine, the algorithm substantially outperforms its closest competitors for problems with few origins and a single destination. It also seems better suited for parallel computation than other shortest path algorithms.

Key words. shortest path, network optimization, auction, parallel algorithms

AMS(MOS) subject classifications. primary 90C47; secondary 90C05

1. Introduction. In this paper we propose a new algorithm for finding shortest paths in a directed graph $(\mathcal{N}, \mathcal{A})$. For the single origin and single destination case, our algorithm is very simple. It maintains a single path starting at the origin. At each iteration, the path is either *extended* by adding a new node, or *contracted* by deleting its terminal node. When the destination becomes the terminal node of the path, the algorithm terminates.

To get an intuitive sense of the algorithm, think of a mouse moving in a graphlike maze, trying to reach a destination. The mouse criss-crosses the maze, either advancing or backtracking along its current path. Each time the mouse backtracks from a node, it records a measure of the desirability of revisiting and advancing from that node in the future (this will be represented by a price variable—see § 2). The mouse revisits and proceeds forward from a node when the node's measure of desirability is judged superior to those of other nodes. Our algorithm efficiently emulates this mouse search process using simple data structures.

In a parallel computing environment, the problem of multiple origins with a single destination can be solved by running in parallel a separate version of the algorithm for each origin. However, the different parallel versions can help each other by sharing the interim results of their computations, thereby substantially enhancing the algorithm's performance. The recent Master's thesis [Pol91] discusses a number of parallel asynchronous implementations of our algorithm, and reports on simulations suggesting a significant speedup potential. Generally, our algorithm seems better suited for parallel computation than all of its competitors.

The practical performance of the algorithm and its numerous variations remain to be fully investigated, particularly using parallel machines. Preliminary experimental results with randomly generated problems on a serial machine, and a comparison with the state-of-the-art shortest path codes of Gallo and Pallotino [GaP88] have been very encouraging. In particular, a code implementing one version of our algorithm outperforms by a large margin its closest competitors for the case of few origins and one

* Received by the editors August 30, 1990; accepted for publication (in revised form) March 28, 1991. This research was supported by National Science Foundation grant DDM-8903385 and by Army Research Office grant DAAL03-86-K-0171.

† Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

destination; see § 7. In a parallel computing environment, the relative advantage of our algorithm should increase, but this remains to be verified in future work.

The worst case running time of the algorithm is pseudopolynomial; it depends on the shortest path lengths. This in itself is not necessarily bad. Dial's algorithm (see [Dia69], [DGK79], [AMO89], [GaP88]) is also pseudopolynomial, yet its running time in practice is excellent, particularly for a small range of arc lengths. Another popular method, the D'Esopo-Pape algorithm [Pap74], has exponential worst case running time [Ker81], [ShW81], yet it performs very well in practice [DGK79], [GaP88]. Nonetheless, under mild conditions, our algorithm can be turned into a polynomial one by using the device of arc length scaling. However, in our computational experiments, this scaling device was entirely unnecessary, and, in fact, degraded the algorithm's performance.

To place our algorithm in perspective, we note that shortest path methods are traditionally divided into two categories: label setting (Dijkstra-like) and label correcting (Bellman-Ford-like); see the surveys given in [AMO89], [GaP86], [GaP88], and the references quoted there. Our algorithm shares features from both types of algorithms. It resembles label setting algorithms in that the shortest distance of a node is found at the first time the node is labeled (becomes the terminal node of the path in our case). It resembles label correcting algorithms in that the label of a node may continue to be updated after its shortest distance is found.

As we explain in § 6, our method may be viewed as a dual coordinate ascent or relaxation method. In reality, the inspiration for the algorithm came from the author's auction and ε-relaxation methods [Ber79], [Ber86] (extensive descriptions of these methods can be found in [Ber88], [BeE88], [BeT89], and [Ber90]). If one applies the ε-relaxation method for a minimum cost flow formulation of the shortest path problem (see § 6), but with the important difference that $\varepsilon = 0$, then one obtains an algorithm which is very similar to the one provided here.

Our algorithm may also be viewed as a special case of the so called *naive auction algorithm*, applied to a special type of assignment problem, which is derived from the shortest path problem (see, e.g., [Law76, p. 186]). The naive auction algorithm, first proposed by Bertsekas in [Ber81] as part of the relaxation method for the assignment problem, and also discussed more recently in the tutorial paper [Ber90], is the same as the auction algorithm, except that the parameter ε that controls the accuracy of the solution is set to zero. The naive auction algorithm is not guaranteed to solve general assignment problems, and is primarily useful as an initialization method for other assignment algorithms, such as relaxation (as described in [Ber81]) or sequential shortest path (as described in [JoV87]). Nevertheless, it is guaranteed to solve the special type of assignment problem, which is relevant to the shortest path context of the present paper.

The paper is organized as follows: In § 2, we describe the basic algorithm for the single origin case and we prove its basic properties. In § 3, we develop the polynomial version of the algorithm using arc length scaling. In § 4, we describe various ways to improve the performance of the algorithm. In § 5, we consider the multiple origin case and we discuss how the algorithm can take advantage of a parallel computing environment. In § 6, we derive the connection with duality and we show that the algorithm may be viewed both as a naive auction algorithm and as a coordinate ascent (or Gauss-Seidel relaxation) method for maximizing a certain dual cost function. Finally, § 7 contains computational results.

2. Algorithm description and analysis. We describe the algorithm in its simplest form for the single origin and single destination case, and we defer the discussion of

other and more efficient versions for subsequent sections. Our main assumption is that *all cycles have positive length*, although we will see shortly that the initialization of the algorithm is greatly simplified if, in addition, all arc lengths are nonnegative.

To simplify the presentation, we will also assume that each node except for the destination has at least one outgoing incident arc; any node not satisfying this condition can be connected to the destination with a very high length arc without materially changing the problem and the subsequent algorithm. We also assume that there is at most one arc between two nodes in each direction, so that we can unambiguously refer to an arc (i, j). Again, this assumption is made for notational convenience; our algorithm can be trivially extended to the case where there are multiple arcs connecting a pair of nodes.

Let node 1 be the origin node and let t be the destination node. In the following, by a *path* we mean a sequence of nodes $(i_1, i_2, \cdots, i_k)$ such that (i_m, i_{m+1}) is an arc for all $m = 1, \cdots, k-1$. If, in addition, the nodes $i_1, i_2, \cdots, i_k$ are distinct, the sequence $(i_1, i_2, \cdots, i_k)$ is called a *simple path*. The length of a path is defined to be the sum of its arc lengths.

The algorithm maintains at all times a simple path $P = (1, i_1, i_2, \cdots, i_k)$. The node i_k is called the *terminal* node of P. The degenerate path $P = (1)$ may also be obtained in the course of the algorithm. If i_{k+1} is a node that does not belong to a path $P = (1, i_1, i_2, \cdots, i_k)$ and (i_k, i_{k+1}) is an arc, *extending P by i_{k+1}* means replacing P by the path $(1, i_1, i_2, \cdots, i_k, i_{k+1})$, called the *extension of P by i_{k+1}*. If P does not consist of just the origin node 1, *contracting* P means replacing P with the path $(1, i_1, i_2, \cdots, i_{k-1})$.

The algorithm also maintains a variable p_i for each node i (called *price* of i) such that

$$p_i \leqq a_{ij} + p_j \quad \forall (i, j) \in \mathcal{A}, \tag{1a}$$

$$p_i = a_{ij} + p_j \quad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } P. \tag{1b}$$

We denote by p the vector of prices p_i. A pair (P, p) consisting of a simple path P and a price vector p that satisfies the above conditions is said to satisfy *complementary slackness* (or CS for short). (When we say that a pair (P, p) satisfies the CS conditions, we implicitly assume that P is simple.)

The CS terminology is motivated by a formulation of the shortest path problem as a linear minimum cost flow problem; see § 6. In this formulation, the prices p_i can be viewed as the variables of a problem which is dual in the usual linear programming duality sense. The complementary slackness conditions for optimality of the primal and dual variables can be shown to be equivalent to the conditions (1). For the moment, however, we ignore the linear programming context, and we simply note that if a pair (P, p) satisfies the CS conditions, then the portion of P between node 1 and any node $i \in P$ is a shortest path from 1 to i, while $p_1 - p_i$ is the corresponding shortest distance. To see this, observe that, by (1b), $p_1 - p_i$ is the length of the portion of P between 1 and i, and by (1a) every path connecting 1 and i must have length at least equal to $p_1 - p_i$.

We will assume that an initial pair (P, p) satisfying CS is available. This is not a restrictive assumption when all arc lengths are nonnegative, since then one can use the default pair

$$P = (1), \quad p_i = 0 \quad \forall i.$$

When some arcs have negative lengths, an initial choice of a pair (P, p) satisfying CS may not be obvious or available, but § 4 provides a general method for finding such a pair.

We now describe the algorithm. Initially, (P, p) is any pair satisfying CS. The algorithm proceeds in iterations, transforming a pair (P, p) satisfying CS into another pair satisfying CS. At each iteration, the path P is either extended by a new node or else is contracted by deleting its terminal node. In the latter case the price of the terminal node is strictly increased. A degenerate case occurs when the path consists by just the origin node 1; in this case the path is either extended, or else is left unchanged with the price p_1 being strictly increased. The iteration is as follows:

TYPICAL ITERATION

Let i be the terminal node of P. If

$$p_i < \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}, \tag{2}$$

go to Step 1; else go to Step 2.

Step 1: (Contract path). Set

$$p_i := \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}, \tag{3}$$

and if $i \neq 1$, contract P. Go to the next iteration.

Step 2: (Extend path). Extend P by node j_i where

$$j_i = \arg\min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}. \tag{4}$$

If j_i is the destination t, stop; P is the desired shortest path. Otherwise, go to the next iteration.

It can be seen that, following the extension Step 2, P is a simple path from 1 to j_i. Indeed, if this were not so, then adding j_i to P would create a cycle, and for every arc (i, j) of this cycle we would have $p_i = a_{ij} + p_j$. Thus, the cycle would have zero length, which is not possible by our assumptions.

Figure 1 provides an example of the operation of the algorithm. In this example, the terminal node traces the tree of shortest paths from the origin to the nodes that are closer to the origin than the given destination. We will see that this behavior is typical when the initial prices are all zero.

PROPOSITION 1. *The pairs (P, p) generated by the algorithm satisfy* CS. *Furthermore, for every pair of nodes i and j, and at all iterations, $p_i - p_j$ is an underestimate of the shortest distance from i to j.*

Proof. We first show by induction that (P, p) satisfies CS. Indeed, the initial pair satisfies CS by assumption. Consider an iteration that starts with a pair (P, p) satisfying CS and produces a pair $(\bar{P}, \bar{p})$. Let i be the terminal node of P. If

$$p_i = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}, \tag{5}$$

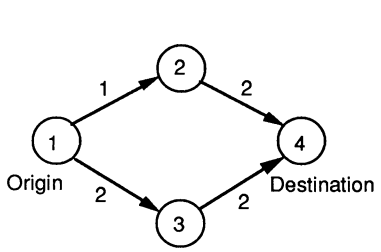
then $\bar{P}$ is the extension of P by a node j_i and $\bar{p} = p$, implying that the CS condition (1b) holds for all arcs of P as well as arc (i, j_i) (since j_i attains the minimum in (5); cf. condition (4)).

Suppose next that

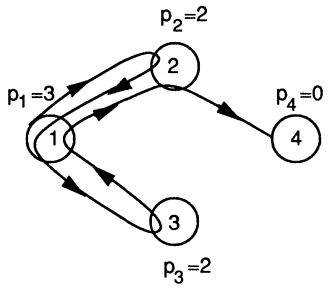
$$p_i < \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}.$$

Then if P is the degenerate path (1), the CS condition holds vacuously. Otherwise, $\bar{P}$ is obtained by contracting P, and for all nodes $j \in \bar{P}$, we have $\bar{p}_j = p_j$, implying conditions (1a) and (1b) for arcs outgoing from nodes of $\bar{P}$. Also, for the terminal node i, we have

$$\bar{p}_i = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$



Shortest path problem with arc lengths as shown



Trajectory of terminal node and final prices generated by the algorithm

Iteration #	Path P prior to the iteration	Price vector p prior to the iteration	Type of action during the iteration
1	(1)	(0, 0, 0, 0)	contraction at 1
2	(1)	(1, 0, 0, 0)	extension to 2
3	(1, 2)	(1, 0, 0, 0)	contraction at 2
4	(1)	(1, 2, 0, 0)	contraction at 1
5	(1)	(2, 2, 0, 0)	extension to 3
6	(1, 3)	(2, 2, 0, 0)	contraction at 3
7	(1)	(2, 2, 2, 0)	contraction at 1
8	(1)	(3, 2, 2, 0)	extension to 2
9	(1, 2)	(3, 2, 2, 0)	extension to 4
10	(1, 2, 4)	(3, 2, 2, 0)	stop

FIG. 1. *An example illustrating the algorithm starting with* $P=(1)$ *and* $p=0$.

implying condition (1a) for arcs outgoing from that node as well. Finally, since $\bar{p}_i > p_i$ and $\bar{p}_k = p_k$ for all $k \neq i$, we have $\bar{p}_k \leqq a_{kj} + \bar{p}_j$ for all arcs (k, j) outgoing from nodes $k \notin P$. This completes the induction proof.

Finally, consider any path from a node i to a node j. By adding the CS condition (1a) along the path, we see that the length of the path is at least $p_i - p_j$, proving the last assertion of the proposition. □

PROPOSITION 2. *If P is a path generated by the algorithm, then P is a shortest path from the origin to the terminal node of P.*

Proof. This follows from the CS property of the pair (P, p) shown in Proposition 1; see the remarks following the CS conditions (1). Furthermore, by the CS condition (1a), every path connecting 1 and i must have length at least equal to $p_1 - p_i$. □

2.1. Interpretation of the algorithm. The preceding propositions can be used to provide an intuitive interpretation of the algorithm. Denote for each node i

$$D_i = \text{shortest distance from the origin 1 to node } i, \tag{6}$$

with $D_1 = 0$ by convention. By Proposition 1, we have, throughout the course of the algorithm,

$$p_1 - p_j \leqq D_j \quad \forall j \in \mathcal{N},$$

while by Proposition 2, we have

$$p_1 - p_i = D_i \quad \text{for all } i \text{ in } P.$$

It follows that

$$D_i + p_i - p_t \leqq D_j + p_j - p_t \quad \forall i \in P \quad \text{and} \quad j \in \mathcal{N}.$$

Since by Proposition 1, $p_i - p_t$ is an estimate of the shortest distance from i to t, we may view the quantity

$$D_j + p_j - p_t$$

as an estimate of the shortest distance from 1 to t using only paths passing through j. Thus, intuitively, it makes sense to consider a node j as "eligible" for inclusion in the algorithm's path only if $D_j + p_j - p_t$ is minimal.

Based on the preceding interpretation, it can be seen that:

(a) The algorithm maintains a path consisting of "eligible" candidates for participation in a shortest path from 1 to t.

(b) The algorithm extends P by a node j if and only if j is an "eligible" candidate.

(c) The algorithm contracts P if the terminal node i has no neighbor which is "eligible." Then, the estimate of i's shortest distance to t is improved (i.e., is increased), and i becomes "ineligible" (since $D_i + p_i - p_t$ is not minimal anymore), thus justifying its deletion from P. Node i will be revisited only after $D_i + p_i - p_t$ becomes minimal again, following sufficiently large increases of the prices of the currently "eligible" nodes.

The preceding interpretation suggests also that the nodes become terminal for the first time in the order of the initial values $D_j + p_j^0 - p_t^0$, where

$$p_i^0 = \text{initial price of node } i. \tag{7}$$

To formulate this property, denote for every node i

$$d_i = D_i + p_i^0. \tag{8}$$

Let us index the iterations by $1, 2, \cdots$, and let

$$k_i = \text{the first iteration index at which node } i \text{ becomes a terminal node,} \tag{9}$$

where, by convention, $k_1 = 0$ and $k_i = \infty$ if i never becomes a terminal node.

PROPOSITION 3. (a) *At the end of iteration k_i we have $p_1 = d_i$.*

(b) *If $k_i < k_j$, then $d_i \leqq d_j$.*

Proof. (a) At the end of iteration k_i, P is a shortest path from 1 to i by Proposition 2, while the length of P is $p_1 - p_i^0$.

(b) By part (a), at the end of iteration k_i, we have $p_1 = d_i$, while at the end of iteration k_j, we have $p_1 = d_j$. Since p_1 is monotonically nondecreasing during the algorithm and $k_i < k_j$, the result follows. □

Note that the preceding proposition shows that when all arc lengths are nonnegative, and the default initialization $p = 0$ is used, the nodes become terminal for the first time in the order of their proximity to the origin.

2.2. Termination and running time of the algorithm. The following proposition establishes the validity of the algorithm.

PROPOSITION 4. *If there exists at least one path from the origin to the destination, the algorithm terminates with a shortest path from the origin to the destination. Otherwise the algorithm never terminates and $p_1 \to \infty$.*

Proof. Assume first that there is a path from node 1 to the destination t. Since by Proposition 1, $p_1 - p_t$ is an underestimate of the (finite) shortest distance from 1 to t, p_1 is monotonically nondecreasing, and p_t is fixed throughout the algorithm, p_1 must stay bounded. We next claim that p_i must stay bounded for all i. Indeed, in order to

have $p_i \to \infty$, node i must become the terminal node of P infinitely often, implying (by Proposition 1) that $p_1 - p_i$ must be equal to the shortest distance from 1 to i infinitely often, which is a contradiction since p_1 is bounded.

We next show that the algorithm terminates finitely. Indeed, it can be seen with a straightforward induction argument that for every node i, p_i is either equal to its initial value, or else it is the length of some path starting at i plus the initial price of the final node of the path; we call this the *modified length* of the path. Every path starting at i can be decomposed into a simple path together with a finite number of cycles, each having positive length by assumption, so the number of distinct modified path lengths within any bounded interval is bounded. Now p_i was shown earlier to be bounded, and each time i becomes the terminal node by extension of the path P, p_i is strictly larger over the preceding time that i became the terminal node of P, corresponding to a strictly larger modified path length. It follows that the number of times i can become a terminal node by extension of the path P is bounded. Since the number of path contractions between two consecutive path extensions is bounded by the number of nodes in the graph, the number of iterations of the algorithm is bounded, implying that the algorithm terminates finitely.

Assume now that there is no path from node 1 to the destination. Then, the algorithm will never terminate, so by the preceding argument, some node i will become the terminal node by extension of the path P infinitely often and $p_i \to \infty$. At the end of iterations where this happens, $p_1 - p_i$ must be equal to the shortest distance from 1 to i, implying that $p_1 \to \infty$. □

We will now estimate the running time of the algorithm, assuming that all the arc lengths and initial prices are integer. Our estimate involves the set of nodes

$$\mathcal{S} = \{i \mid d_i \leqq d_t\}; \tag{10}$$

by Proposition 3, these are the only nodes that ever become terminal nodes of the paths generated by the algorithm. Let us denote

$$I = \text{number of nodes in } \mathcal{S}, \tag{11}$$

$$G = \text{maximum out-degree (number of outgoing arcs) over the nodes in } \mathcal{S}, \tag{12}$$

and let us also denote by E the product

$$E = I \cdot G. \tag{13}$$

PROPOSITION 5. *Assume that there exists at least one path from the origin* 1 *to the destination t, and that the arc lengths and initial prices are all integer. The worst case running time of the algorithm is* $O(E(D_t + p_t^0 - p_1^0))$.

Proof. Each time a node i becomes the terminal node of the path, we have $p_i = p_1 - D_i$ (cf. Proposition 2). Since at all times we have $p_1 \leqq D_t + p_t^0$ (cf. Proposition 1), it follows that

$$p_i = p_1 - D_i \leqq D_t + p_t^0 - D_i,$$

and using the definitions $d_t = D_t + p_t^0$ and $d_i = D_i + p_i^0$, and the fact $d_i \geqq d_1$ (cf. Proposition 3), we see that throughout the algorithm, we have

$$p_i - p_i^0 \leqq d_t - d_i \leqq d_t - d_1 = D_t + p_t^0 - p_1^0 \quad \forall i \in \mathcal{S}. \tag{14}$$

Therefore, since prices increase by integer amounts, $D_t + p_t^0 - p_1^0 + 1$ bounds the number of times that p_i increases (with an attendant path contraction if $i \neq 1$). Since the computation per iteration is bounded by a constant multiple of the out-degree of the terminal node of the path, we see that the computation corresponding to contractions and price increases is $O(E(D_t + p_t^0 - p_1^0))$.

The number of path extensions with $i \in \mathcal{I}$ becoming the terminal node of the path is bounded by the number of increases of p_i, which in turn is bounded by $D_t + p_t^0 - p_1^0 + 1$. Thus the computation corresponding to extensions is also $O(E(D_t + p_t^0 - p_1^0))$. □

Note that we have $D_t \leqq hL$, where

$$L = \max_{(i,j)\in\mathcal{A}} a_{ij}, \tag{15}$$

$$h = \text{minimum number of arcs in a shortest path from 1 to } t. \tag{16}$$

Then in the special case where all arc lengths are nonnegative, and for the default price vector $p = 0$, Proposition 5 yields the running time estimate

$$O(EhL). \tag{17}$$

As the preceding estimate suggests, the running time can depend on L, as illustrated in Fig. 2 for a graph involving a cycle with relatively small length. This is the same type of graph for which the Bellman-Ford method starting with the zero initial conditions performs poorly (see [BeT89, p. 298]).

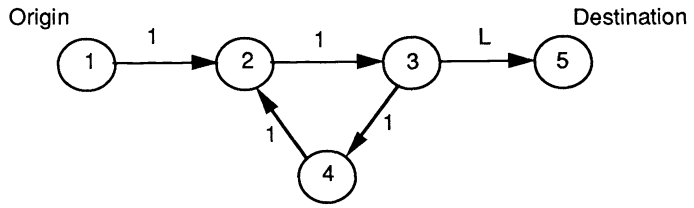


FIG. 2. *Example graph for which the number of iterations of the algorithm is not polynomially bounded. The lengths are shown next to the arcs and $L > 1$. By tracing the steps of the algorithm starting with $P = (1)$ and $p = 0$, we see that the price of node 3 will be first increased by 1 and then it will be increased by increments of 3 (the length of the cycle) as many times as necessary for p_3 to reach L.*

In the next section we will modify the algorithm to improve its complexity. However, we believe that the estimate of Proposition 5 is far from representative of the algorithm's "average" performance. For randomly generated problems, it appears that the number of iterations can be estimated quite reliably (within a constant factor roughly equal to two) by

$$n_t - 1 + \sum_{i\in\mathcal{I}, i\neq t} (2n_i - 1),$$

where n_i is the number of nodes in a shortest path from 1 to i; for example, for the problem of Fig. 1, the above estimate is exact.

2.3. The case of multiple destinations. We finally note that when there is a single origin and multiple destinations, the algorithm can be applied with virtually no change. We simply stop the algorithm when all destinations have become the terminal node of the path P at least once. If, initially, we choose $p_i = 0$ for all i, the destinations will be reached in the order of their proximity to the origin, as shown by Proposition 3. We also note that the algorithm can be similarly applied to a problem with multiple origins and a single destination, by first reversing the roles of origins and destinations, and the direction of each arc.

3. Arc length scaling. Throughout this section (and only this section) we will assume that all arc lengths are nonnegative. We introduce a version of the algorithm

where the shortest path problem is solved several times, each time with different arc lengths and starting prices. Let

$$K = \lfloor \log L \rfloor + 1 \tag{18}$$

and for $k = 1, \cdots, K$, define

$$a_{ij}(k) = \left\lfloor \frac{a^{ij}}{2^{K-k}} \right\rfloor \quad \forall (i,j) \in \mathcal{A}. \tag{19}$$

Note that $a_{ij}(k)$ is the integer consisting of the k most significant bits in the K-bit binary representation of a_{ij}. Define

$$\bar{k} = \min \{k \geqq 1 \mid \text{each cycle has positive length}\}. \tag{20}$$

The following algorithm is predicated on the assumption that $\bar{k}$ is a small integer that does not grow beyond a certain bound as K increases. This is true for many problem types; for example, when the graph is acyclic, in which case $\bar{k} = 1$. For the case where this is not so, a slightly different arc length scaling procedure can be used; see the next section.

The scaled version of the algorithm solves $K - \bar{k} + 1$ shortest path problems, called *subproblems.* The arc lengths for subproblem k, $k = \bar{k}, \cdots, K$, are $a_{ij}(k)$ and the starting prices are obtained by doubling the final prices $p_i^*(k)$ of the previous subproblem

$$p_i^0(k+1) = 2p_i^*(k) \quad \forall i \in \mathcal{N}, \tag{21}$$

except for the first subproblem ($k = \bar{k}$), where we take

$$p_i^0(\bar{k}) = 0 \quad \forall i \in \mathcal{N}.$$

Note that we have $a_{ij}(K) = a_{ij}$ for all (i,j), and the last subproblem is equivalent to the original. Since the length of a cycle with respect to arc lengths $a_{ij}(\bar{k})$ is positive (by the definition of $\bar{k}$) and from the definition (19), we have

$$0 \leqq a_{ij}(k+1) - 2a_{ij}(k) \leqq 1 \quad \forall (i,j) \in \mathcal{A}, \tag{22}$$

it follows that cycles have positive length for each subproblem. Furthermore, in view of (22) and the doubling of the prices at the end of each subproblem (cf. (19)), the CS condition

$$p_i^0(k+1) \leqq p_j^0(k+1) + a_{ij}(k+1) \quad \forall (i,j) \in \mathcal{A} \tag{23}$$

is satisfied at the start of subproblem $k+1$, since it is satisfied by $p_i^*(k)$ at the end of subproblem k. Therefore, the algorithm of the preceding section can be used to solve all the subproblems.

Let $D_t(k)$ be the shortest distance from 1 to t for subproblem k and let

(24) $h(k) =$ the number of arcs in the final path from 1 to t in subproblem k.

It can be seen using (22) that

$$D_t(k+1) \leqq 2D_t(k) + h(k),$$

and in view of (21), we obtain

$$D_t(k+1) \leqq 2(p_1^*(k) - p_t^*(k)) + h(k) = p_1^0(k+1) - p_t^0(k+1) + h(k).$$

Using Proposition 5, it follows that the running time of the algorithm for subproblem k, $k = \bar{k}+1, \cdots, K$, is

$$O(E(k)h(k)), \tag{25}$$

where $E(k)$ is the number of the form (12) corresponding to subproblem k. The running time of the algorithm for subproblem $\bar{k}$ is

$$O(E(\bar{k})D_t(\bar{k})), \tag{26}$$

where $D_t(\bar{k})$ is the shortest distance from 1 to t corresponding to the lengths $a_{ij}(\bar{k})$. Since

$$a_{ij}(\bar{k}) < 2^{\bar{k}},$$

we have

$$D_t(\bar{k}) < 2^{\bar{k}} h(\bar{k}). \tag{27}$$

Adding over all $k = \bar{k}, \cdots, K$, we see that the running time of the scaled version of the algorithm is

$$O\left(2^{\bar{k}} E(\bar{k})h(\bar{k}) + \sum_{k=\bar{k}+1}^{K} E(k)h(k)\right). \tag{28}$$

Assuming that $\bar{k}$ is bounded as L increases, the above expression is bounded by $O(NG\bar{h} \log L)$, where $\bar{h} = \max_{k=\bar{k},\cdots,K} h(k)$, N is the number of nodes, and G is the maximum out-degree of a node. These worst-case estimates of running time are still inferior to the sharpest estimate $O(A + N \log N)$ available for implementations of Dijkstra's method, where A is the number of arcs. The estimate (28) compares favorably with the estimate $O(Ah)$ for the Bellman-Ford algorithm when $2^{\bar{k}} \max_k E(k)$ is much smaller than A; this may occur if the destination is close to the origin relative to other nodes, in which case $\max_k E(k)$ may be much smaller than A.

We finally note that we can implement arc length scaling without knowing the value of $\bar{k}$. We can simply guess an initial value of $\bar{k}$, say $\bar{k} = 1$, apply the algorithm for lengths $a_{ij}(\bar{k})$, and at each path extension, check whether a cycle is formed. If so, we increment $\bar{k}$, we double the current prices, we reset the path to $P = (1)$, and we restart the algorithm with the new data and initial conditions. Eventually, after a finite number of restarts, we will obtain a value of $\bar{k}$ which is large enough for cycles never to form during the rest of the algorithm. The computation done up to that point, however, will not be entirely wasted; it will serve to provide a better set of initial prices.

4. Efficient implementation, two-sided algorithm, and preprocessing. The main computational bottleneck of the algorithm is the calculation of $\min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}$, which is done every time node i becomes the terminal node of the path. We can reduce the number of these calculations using the following observation. Since the CS condition (1a) is maintained at all times, if some arc (i, j_i) satisfies

$$p_i = a_{ij_i} + p_{j_i},$$

it follows that

$$a_{ij_i} + p_{j_i} = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$

so the path can be extended by j_i if i is the terminal node of the path. This suggests the following implementation strategy: each time a path contraction occurs with i being the terminal node, we calculate

$$\min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\},$$

together with an arc (i, j_i) such that

$$j_i = \arg\min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}.$$

At the next time node i becomes the terminal node of the path, we check whether the condition $p_i = a_{ij_i} + p_{j_i}$ is satisfied, and if so, we extend the path by node j_i without going through the calculation of $\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}$. In practice, this device is very effective, typically saving from a third to a half of the calculations of the preceding expression. The reason is that the test $p_i = a_{ij_i} + p_{j_i}$ is rarely failed; the only way it can fail is when the price p_{j_i} is increased between the two successive times i became the terminal node of the path.

The preceding idea can be strengthened further. Suppose that whenever we compute the "best neighbor"

$$j_i = \arg\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\},$$

we also compute the "second best neighbor" k_i given by

$$k_i = \arg\min_{(i,j)\in\mathcal{A},\, j\neq j_i}\{a_{ij}+p_j\},$$

and the corresponding "second best level"

$$w_i = a_{ik_i} + p_{k_i}.$$

Then, at the next time node i becomes the terminal node of the path, we can check whether the condition $a_{ij_i} + p_{j_i} \leqq w_i$ is satisfied, and if so, we know that j_i still attains the minimum in the expression

$$\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\},$$

thereby obviating the calculation of this minimum. If on the other hand we have $a_{ij_i} + p_{j_i} > w_i$ (due to an increase of p_{j_i} subsequent to the calculation of w_i), we can check to see whether we still have $w_i = a_{ik_i} + p_{k_i}$; if this is so, then k_i becomes the "best neighbor,"

$$k_i = \arg\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\},$$

thus obviating again the calculation of the minimum.

With proper implementation, the devices outlined above can typically reduce the number of calculations of the expression $\min_{(i,j)\in\mathcal{A}}\{a_{ij}+p_j\}$ by a factor in the order of three to five, thereby dramatically reducing the total computation time.

4.1. The two-sided algorithm. In shortest path problems, one can exchange the role of origins and destinations by reversing the direction of all arcs. It is therefore possible to use a destination-oriented version of our algorithm which maintains a path R that *ends* at the destination and changes at each iteration by means of a contraction or an extension. This algorithm, presented below and called the *reverse algorithm*, is equivalent to the algorithm in § 2, which will henceforth be referred to as the *forward algorithm*. The CS conditions for the problem with arc directions reversed are

$$\bar{p}_j \leqq a_{ij} + \bar{p}_i \quad \forall (i,j)\in\mathcal{A},$$

$$\bar{p}_j = a_{ij} + \bar{p}_i \quad \text{for all pairs of successive nodes } i \text{ and } j \text{ of } R,$$

where $\bar{p}$ is the price vector. By replacing $\bar{p}$ by $-p$, we obtain the CS conditions in the form of (1), thus maintaining a common CS condition for both the forward and the reverse algorithm. The following description of the reverse algorithm also replaces $\bar{p}$ by $-p$, with the result that the prices are *decreasing* instead of increasing. To be consistent with the assumptions made regarding the forward algorithm, we assume that each node except for the origin has at least one incoming arc.

In the reverse algorithm, initially, R is any path ending at the destination and p is any price vector satisfying the CS conditions (1) together with R; for example,

$$R=(t), \qquad p_i=0 \quad \forall i,$$

if all arc lengths are nonnegative.

TYPICAL ITERATION OF THE REVERSE ALGORITHM

Let j be the starting node of R. If

$$p_j > \max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\},$$

go to Step 1; else go to Step 2.

Step 1: (Contract path). Set

$$p_j := \max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\},$$

and if $j \neq t$, contract R (that is, delete the starting node j of R). Go to the next iteration.

Step 2: (Extend path). Extend R by node i_j, (that is, make i_j the starting node of R, preceding j), where

$$i_j = \arg\max_{(i,j)\in\mathcal{A}} \{p_i - a_{ij}\}.$$

If i_j is the origin 1, stop; R is the desired shortest path. Otherwise, go to the next iteration.

The reverse algorithm is really the forward algorithm applied to a reverse shortest path problem, so by the results of § 2, it is valid and obtains a shortest path in a finite number of iterations, assuming that at least one path exists from 1 to t.

We now consider combining the forward and the reverse algorithms into one. In this combined algorithm, we initially have a price vector p and two paths P and R satisfying CS together with p, where P starts at the origin and R ends at the destination. The paths P and R are extended and contracted according to the rules of the forward and the reverse algorithms, respectively, and the combined algorithm terminates when P and R have a common node. Both P and R satisfy CS together with p throughout the algorithm, so when P and R meet, say at node i, the composite path consisting of the portion of P from 1 to i and the portion of R from i to t will be shortest.

COMBINED ALGORITHM

Step 1: (Run forward algorithm). Execute several iterations of the forward algorithm (subject to the termination condition), at least one of which leads to an increase of the origin price p_1. Go to Step 2.

Step 2: (Run reverse algorithm). Execute several iterations of the reverse algorithm (subject to the termination condition), at least one of which leads to a decrease of the destination price p_t. Go to Step 1.

To justify the combined algorithm, note that p_1 can only increase and p_t can only decrease during its course, while the difference $p_1 - p_t$ can be no more than the shortest distance between 1 and t. Assume that the arc lengths and the initial prices are integer, and that there is at least one path from 1 to t. Then, p_1 and p_t can only change by integer amounts and $p_1 - p_t$ is bounded. Hence, p_1 and p_t can change only a finite number of times, guaranteeing that there will be only a finite number of executions of Steps 1 and 2 of the combined algorithm. By the results of § 2, each Step 1 and Step 2 must contain only a finite number of iterations of the forward and the reverse

algorithms, respectively. It follows that the algorithm must terminate in a finite number of iterations. Note that this argument relies on the requirement that p_1 increases at least once in Step 1 and p_t decreases at least once in Step 2. Without this requirement, one can construct examples showing that the combined algorithm may never terminate. Note also that our termination proof depends on the problem data being integer. For real problem data, we have been unable to prove termination or to disprove it with a counterexample.

One motivation for the combined algorithm is that two processors can be used in parallel to maintain the forward and the reverse paths while sharing the same price vector. However, there is another motivation. Based on our computational results, the combined algorithm is much faster than both the forward and the reverse algorithms.

4.2. Initialization and preprocessing. In order to initialize the algorithm, one should have a price vector p satisfying $p_i \leqq a_{ij} + p_j$ for all arcs (i, j). When some arc lengths are negative, the default choice $p = 0$ does not satisfy this condition, and there may be no obvious initial choice for p. In other situations, even when all arc lengths are nonnegative, it may be preferable to use a "favorable" initial price vector in place of the default choice $p = 0$. This possibility arises in a reoptimization context with slightly different arc length data, or with a different origin and/or destination. However, the "favorable" initial price vector may not satisfy the preceding condition.

To cope with situations such as the above, we provide a *preprocessing algorithm* for obtaining an appropriate initial vector p satisfying the condition $p_i \leqq a_{ij} + p_j$ for all arcs (i, j) (except for the immaterial outgoing arcs from the destination t).

To be precise, suppose that we have a vector $\bar{p}$, which, together with a set of arc lengths $\{\bar{a}_{ij}\}$, satisfies $\bar{p}_i \leqq \bar{a}_{ij} + \bar{p}_j$ for all arcs (i, j), and that we are given a new set of arc lengths $\{a_{ij}\}$. We describe a preprocessing algorithm for obtaining a vector p satisfying $p_i \leqq a_{ij} + p_j$ for all arcs (i, j). (Thus, to deal with the case where some arc lengths are negative and no appropriate initial vector is known, one can take $\bar{p} = 0$ and $\bar{a}_{ij} = \max\{0, a_{ij}\}$.) The algorithm maintains a subset of arcs $\mathscr{E}$ and a price vector p. Initially,

$$\mathscr{E} = \{(i, j) \in \mathscr{A} \mid a_{ij} < \bar{a}_{ij},\ i \neq t\}, \qquad p = \bar{p}.$$

The typical iteration is as follows:

TYPICAL PREPROCESSING ITERATION

Step 1: (Select arc to scan). If $\mathscr{E}$ is empty, stop; otherwise, remove an arc (i, j) from $\mathscr{E}$ and go to Step 2.

Step 2: (Add affected arcs to $\mathscr{E}$). If $p_i > a_{ij} + p_j$, set

$$p_i := a_{ij} + p_j$$

and add to $\mathscr{E}$ every arc (k, i) with $k \neq t$ that does not already belong to $\mathscr{E}$.

We have the following proposition.

PROPOSITION 6. *Assume that each node i is connected to the destination t with at least one path. Then the preprocessing algorithm terminates in a finite number of iterations with a price vector p satisfying*

$$p_i \leqq a_{ij} + p_j \quad \forall (i, j) \in \mathscr{A} \quad \text{with } i \neq t. \tag{29}$$

Proof. We first note that by induction we can prove that throughout the algorithm we have

$$\mathscr{E} \supset \{(i, j) \in \mathscr{A} \mid p_i > a_{ij} + p_j,\ i \neq t\}.$$

As a result, when $\mathscr{E}$ becomes empty, the condition (29) is satisfied. Next, observe that by induction it can be seen that throughout the algorithm, p_i is equal to the modified length of some path starting at i (the length of the path plus the initial price of the final node of the path; see the proof of Proposition 4). Thus, termination of the algorithm will follow as in the proof of Proposition 4 (using the fact that cycle lengths are positive and prices are monotonically nonincreasing throughout the algorithm), provided we can show that the prices are bounded from below. Indeed, let

$$p_k^* = \begin{cases} \bar{p}_t + \text{shortest distance from } k \text{ to } t & \text{if } k \neq t, \\ \bar{p}_t & \text{if } k = t, \end{cases}$$

and let r be a sufficiently large scalar so that

$$\bar{p}_k \geqq p_k^* - r \quad \forall k.$$

We show by induction that throughout the algorithm we have

$$p_k \geqq p_k^* - r \quad \forall k \neq t. \tag{30}$$

Indeed, this condition holds initially by the choice of r. Suppose that the condition holds at the start of an iteration where arc (i, j) with $i \neq t$ is removed from $\mathscr{E}$. We then have

$$a_{ij} + p_j \geqq a_{ij} + p_j^* - r \geqq \min_{(i,m)\in\mathscr{A}} \{a_{im} + p_m^*\} - r = p_i^* - r,$$

where the last equality holds in view of the definition of p_k^* as a constant plus the shortest distance from k to t. Therefore, the iteration preserves the condition (30) and the prices p_i remain bounded throughout the preprocessing algorithm. This completes the proof. □

If the new arc lengths differ from the old ones by "small" amounts, it can be reasonably expected that the preprocessing algorithm will terminate quickly. This hypothesis, however, must be tested empirically on a problem-by-problem basis.

In the preceding preprocessing iteration, node prices can only decrease. An alternative iteration where node prices can only increase starts with

$$\mathscr{E} = \{(i, j) \in \mathscr{A} \mid a_{ij} < \bar{a}_{ij}, j \neq 1\}, \qquad p = \bar{p}.$$

and operates as follows:

ALTERNATIVE PREPROCESSING ITERATION

Step 1: (Select arc to scan). If $\mathscr{E}$ is empty, stop; otherwise, remove an arc (i, j) from $\mathscr{E}$ and go to Step 2.

Step 2: (Add affected arcs to $\mathscr{E}$). If $p_i > a_{ij} + p_j$, set

$$p_j := p_i - a_{ij}$$

and add to $\mathscr{E}$ every arc (j, k) with $k \neq 1$ that does not already belong to $\mathscr{E}$.

This algorithm is the preceding preprocessing algorithm (where prices decrease monotonically), but is applied to the reverse shortest path problem, where the arc directions have been reversed and the roles of origin and destination have been exchanged (cf. the two-sided algorithm given earlier). The following proposition therefore follows from Proposition 6.

PROPOSITION 7. *Assume that the origin node* 1 *is connected to each node i with at least one path. Then the alternative preprocessing algorithm terminates in a finite number of iterations with a price vector p satisfying*

$$p_i \leqq a_{ij} + p_j \quad \forall (i, j) \in \mathscr{A} \quad \textit{with } j \neq 1.$$

The preprocessing idea can also be used in conjunction with arc length scaling in the case where the integer $\bar{k}$ of (20) is large or unknown. We can then use, in place of the scaled arc lengths $a_{ij}(k)$ of (19), the arc lengths

$$\tilde{a}_{ij}(k) = \left\lceil \frac{a_{ij}}{2^{K-k}} \right\rceil \quad \forall (i,j) \in \mathcal{A},$$

in which case we will have $\tilde{a}_{ij}(k) > 0$ if $a_{ij} > 0$. As a result, every cycle will have positive length with respect to arc lengths $\{\tilde{a}_{ij}(k)\}$ for all k. The difficulty now, however, is that (22) and (23) may not be satisfied. In particular, we will have instead

$$-1 \leqq \tilde{a}_{ij}(k+1) - 2\tilde{a}_{ij}(k) \leqq 0 \quad \forall (i,j) \in \mathcal{A},$$

and

$$p_i^0(k+1) \leqq p_j^0(k+1) + \tilde{a}_{ij}(k+1) + 1 \quad \forall (i,j) \in \mathcal{A}, \tag{31}$$

and the vector $p^0(k+1)$ may not satisfy the CS conditions with respect to arc lengths $\{\tilde{a}_{ij}(k+1)\}$. The small violation of the CS conditions indicated in (31) can be rectified by applying the preprocessing algorithm at the beginning of each subproblem. It is then possible to prove a polynomial complexity bound for the corresponding arc length scaling algorithm, by proving a polynomial complexity bound for the preprocessing algorithm and by using very similar arguments to those used in the previous section.

5. Parallelization issues. When there is a single destination and multiple origins, several interesting parallel computation possibilities arise. The idea is to maintain a different path P^i for each origin i, and possibly, a reverse path R for the destination. Different paths may be handled by different processors, and price information can be shared by the processors in some way. There are several possible implementations of this idea. We will describe two of these implementations, motivated by the architectures of shared memory and message passing machines, respectively. For simplicity, we will not consider the possibility of using the reverse path R. In [Pol91], Polymenakos discusses parallel two-sided algorithms.

5.1. Shared memory implementation. Here, there is a common price vector p stored in memory that is accessible by all processors. For each origin i, there is a path P^i satisfying CS together with p. In a synchronous implementation of the algorithm, an iteration is executed simultaneously for some origins (possibly all origins, depending on the availability of processors). At the end of an iteration, the results corresponding to the different origins are coordinated. To this end, we note that if a node is the terminal node of the path of several origins, the result of the iteration will be the same for all these origins, i.e., a path extension or a path contraction and corresponding price change will occur simultaneously for all these origins. The only potential conflict arises when a node i is the terminal path node for some origin and the path of a different origin is extended by i as a result of the iteration. Then, if p_i is increased due to a path contraction for the former origin, the path extension of the latter origin is cancelled. An additional important detail is that an origin i can stop its computation once the terminal node of its path P^i is an origin that has already found its shortest path to the destination. Thus, the processor handling this origin may be diverted to handle the path of another origin.

It is reasonable to speculate that the parallel time to solve the multiple origins problem is closer to the smallest time over all origins to find a single origin shortest path, rather than to the longest time. However, this conjecture needs to be tested experimentally on a shared memory machine.

The parallel implementation outlined above is synchronous, that is, all origins iterate simultaneously, and the results are communicated and coordinated at the end of the iteration to the extent necessary for the next iteration. An asynchronous implementation is also possible, principally because of the monotonicity of the mapping

$$p_i := \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\};$$

see [Ber82] and [BeT89]. We refer to [Pol91] for a discussion of such an asynchronous implementation.

5.2. Message passing implementation. Here, for each origin i, there is a separate processor that executes the forward algorithm and keeps in local memory a price vector p^i and a corresponding path P^i satisfying CS together with p^i. The price vectors are communicated at various times to other processors, perhaps irregularly. A processor operating on (P^i, p^i), upon reception of a price vector p^j from another processor j, adopts as the price of each node n the *maximum* of the prices of n according to the existing and the received price vectors, that is,

$$p_n^i := \max\{p_n^i, p_n^j\} \quad \forall n \in \mathcal{N}. \tag{32}$$

The processor also uses the updated price vector p^i to delete successively, starting with the terminal node, the arcs (m, n) of P^i for which the equality $p_m = a_{mn} + p_n$ is violated. The CS property is maintained in this way because it can be shown that the updated price vector p^i satisfies the condition

$$p_m^i \leqq a_{mn} + p_n^i \quad \forall (m, n) \in \mathcal{A}.$$

This is the subject of the following proposition.

PROPOSITION 8. *Let p^i and p^j be two price vectors satisfying*

$$p_m^i \leqq a_{mn} + p_n^i, \qquad p_m^j \leqq a_{mn} + p_n^j \quad \forall (m, n) \in \mathcal{A}. \tag{33}$$

Then,

$$\max\{p_m^i, p_m^j\} \leqq a_{mn} + \max\{p_n^i, p_n^j\} \quad \forall (m, n) \in \mathcal{A}, \tag{34}$$

and

$$\min\{p_m^i, p_m^j\} \leqq a_{mn} + \min\{p_n^i, p_n^j\} \quad \forall (m, n) \in \mathcal{A}. \tag{35}$$

Proof. From (33), we have

$$p_m^i \leqq a_{mn} + \max\{p_n^i, p_n^j\} \quad \forall (m, n) \in \mathcal{A},$$

and

$$p_m^j \leqq a_{mn} + \max\{p_n^i, p_n^j\} \quad \forall (m, n) \in \mathcal{A}.$$

Combining these two relations, we obtain (34). The proof of (35) is similar. □

Note that even with no communication between the processors, the algorithm would still involve considerable parallelism, since a multiple origin problem would be solved in the time needed to solve a single origin problem. Combining the price vectors of several processors, however, tends to speed up the termination of the algorithm for all origins. In fact, if there are more processors than origins, it may still be beneficial to create some additional artificial origins in order to obtain additional price vectors. The drawback of this implementation is that communication of the price vectors may be relatively slow, and that combining two price vectors according to (32) may be time-consuming if no vector processing hardware is available at the processors.

6. Relation to naive auction and dual coordinate ascent. We now explain how our (forward) single origin-single destination algorithm can be viewed as an instance of the application of the naive auction algorithm to a special type of assignment problem.

The naive auction algorithm is applicable to assignment problems where we have to match n persons and n objects on a one-to-one basis. There is a cost c_{ij} for matching person i with object j and we want to assign persons to objects so as to minimize the total cost. There is also a restriction that person i can be assigned to object j only if (i,j) belongs to a set of given pairs $\mathcal{A}$. Mathematically, we want to find a feasible assignment that minimizes the total cost $\sum_{i=1}^{n} c_{ij_i}$, where by a feasible assignment we mean a set of person-object pairs $(1,j_1),\cdots,(n,j_n)$, such that the objects $j_1,\cdots,j_n$ are all distinct and $(i,j_i)\in\mathcal{A}$ for all i. (Auction algorithms are usually described in terms of maximization of the total "benefit" of the assignment; see, for example, [Ber90]. It is, however, convenient here to reformulate the problem and the algorithm in terms of minimization; this amounts to reversing the signs of the cost coefficients and the prices, and replacing maximization by minimization.)

The naive auction algorithm proceeds in iterations and generates a sequence of price vectors p and partial assignments (that is, assignments where only a subset of the persons have been matched with objects). At the beginning of each iteration, the condition

$$c_{ij_i}+p_{j_i}=\min_{(i,j)\in\mathcal{A}}\{c_{ij}+p_j\} \tag{36}$$

is satisfied for all pairs (i,j_i) of the partial assignment. The initial price vector–partial assignment pair is required to satisfy this condition, but is otherwise arbitrary. If all persons are assigned, the algorithm terminates. If not, some person who is unassigned, say i, is selected. This person finds an object j_i, which is best in the following sense:

$$j_i\in\arg\min_{(i,j)\in\mathcal{A}}\{c_{ij}+p_j\};$$

and then:

(a) Gets assigned to the best object j_i; the person that was assigned to j_i at the beginning of the iteration (if any) becomes unassigned.

(b) Sets the price of j_i to the level at which he/she is indifferent between j_i and the second best object, that is, he/she sets p_{j_i} to

$$p_{j_i}+w_i-v_i,$$

where v_i is the cost for acquiring the best object (including payment of the corresponding price),

$$v_i=\min_{(i,j)\in\mathcal{A}}\{c_{ij}+p_j\},$$

and w_i is the cost for acquiring the second best object

$$w_i=\min_{(i,j)\in\mathcal{A},j\neq j_i}\{c_{ij}+p_j\}.$$

This process is repeated in a sequence of iterations until each person is assigned to an object.

The naive auction algorithm differs from the auction algorithm in the choice of the price increase increment. In the auction algorithm the price p_{j_i} is increased by $w_i-v_i+\varepsilon$, where ε is a small positive constant. Thus the naive auction algorithm is the same as the auction algorithm, except that $\varepsilon=0$. This is, however, a significant difference; while the auction algorithm is guaranteed to terminate in a finite number

of iterations if at least one feasible assignment exists, the naive auction algorithm may cycle indefinitely, with some objects remaining unassigned. If, however, the naive auction algorithm terminates, the feasible assignment obtained upon termination is optimal. The reason is that (36) may be viewed as a complementary slackness condition for the linear programming problem associated with the assignment problem, and by a classical linear programming result, this condition, together with feasibility, guarantees optimality of the final assignment.

6.1. Formulation of the shortest path problem as an assignment problem. Now, given the shortest path problem described in § 2, with node 1 as origin and node t as destination, we formulate the following assignment problem.

Let $2, \cdots, N$ be the "object" nodes, and for each node $i \neq t$, introduce a "person" node i'. For every arc (i, j) of the shortest path problem with $i \neq t$ and $j \neq 1$, introduce the arc (i', j) with cost a_{ij} in the assignment problem. Also introduce the zero cost arc (i', i) for each $i \neq 1, t$. Figure 3 illustrates the assignment problem.

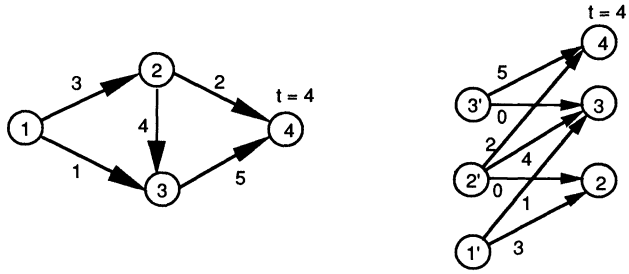


FIG. 3. *A shortest path problem and its corresponding assignment problem. The arc lengths and the assignment costs are shown next to the arcs.*

Now consider applying the naive auction algorithm starting from a price vector p satisfying the CS condition (1a), i.e.,

$$p_i \leqq a_{ij} + p_j \quad \forall (i, j) \in \mathcal{A}, \tag{37}$$

and the partial assignment

$$(i', i) \quad \forall i \neq 1, t.$$

This initial pair satisfies the corresponding condition (36), because the cost of the assigned arcs (i', i) is zero.

We impose an additional rule for breaking ties in the naive auction algorithm: if at some iteration involving the unassigned person i', the arc (i', i) is the best arc and is equally desirable with some other arc (i', j_i) (i.e., $p_i = a_{ij_i} + p_{j_i} = \min_{(i,j)\in\mathcal{A}} \{a_{ij} + p_j\}$), then the latter arc is preferred; that is, (i', j_i) is added to the assignment rather than (i', i). Furthermore, we introduce an inconsequential modification of the naive auction iteration involving a bid of person $1'$, in order to account for the special way of handling a contraction at the origin in the shortest path algorithm. In particular, the bid of $1'$ will consist of finding an object j_1 attaining the minimum in

$$\min_{(1,j)\in\mathcal{A}} \{a_{1j} + p_j\},$$

assigning j_1 to $1'$, and deassigning the person assigned to j_1 (in the case $j_1 \neq t$), but *not* changing the price p_{j_1}.

It can now be shown that the naive auction algorithm under the preceding conditions is equivalent to the (forward) shortest path algorithm of § 2. In particular, the following can be verified by induction:

(a) The CS condition (37) is preserved by the naive auction algorithm.

(b) Each assignment generated by the algorithm consists of a sequence of the form

$$(1', i_1), (i_1', i_2), \cdots, (i_{k-1}', i_k), \tag{38}$$

together with the additional arcs

$$(i', i) \quad \text{for } i \neq i_1, \cdots, i_k, t,$$

and corresponds to a path $P = (1, i_1, \cdots, i_k)$ generated by the shortest path algorithm. As long as $i_k \neq t$, the (unique) unassigned person in the naive auction algorithm is person i_k', corresponding to the terminal node of the path. When $i_k = t$, a feasible assignment results, in which case the naive auction algorithm terminates, consistently with the termination criterion for the shortest path algorithm.

(c) In an iteration corresponding to an unassigned person i' with $i \neq 1$, the arc (i', i) is always a best arc; this is a consequence of the complementary slackness condition (37). Furthermore, there are three possibilities: (1) (i', i) is the unique best arc, in which case (i', i) is added to the assignment, and the price p_i is increased by

$$\min_{(i,j)\in\mathcal{A}} \{c_{ij} + p_j\} - p_i;$$

this corresponds to contracting the current path by the terminal node i. (2) There is an arc (i', j_i) with $j_i \neq t$, which is equally preferred to (i', i), that is,

$$p_i = a_{ij_i} + p_{j_i},$$

in which case, in view of the tie-breaking rule specified earlier, (i', j_i) is added to the assignment and the price p_{j_i} remains the same. Furthermore, the object j_i must have been assigned to j_i' at the start of the iteration, so adding (i', j_i) to the assignment (and removing (j_i', j_i)) corresponds to extending the current path by node j_i. (The positivity assumption on the cycle lengths is crucial for this property to hold.) (3) The arc (i', t) is equally preferred to (i', i), in which case the heretofore unassigned object t is assigned to i', thereby terminating the naive auction algorithm; this corresponds to the destination t becoming the terminal node of the current path, thereby terminating the shortest path algorithm.

We have thus seen that the shortest path algorithm may be viewed as an instance of the naive auction algorithm. However, the properties of the former algorithm do not follow from generic properties of the latter. As mentioned earlier, the naive auction algorithm need not terminate, in general. In the present context, it does terminate thanks to the special structure of the corresponding assignment problem, and also thanks to the positivity assumption on all cycle lengths.

6.2. Relation to dual coordinate ascent. We next explain how the single origin-single destination algorithm can be viewed as a dual coordinate ascent method. The shortest path problem can be written in the minimum cost flow format

$$\text{(LNF)} \qquad \text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} a_{ij}x_{ij}$$

$$\text{subject to} \quad \sum_{\{j|(i,j)\in\mathcal{A}\}} x_{ij} - \sum_{\{j|(j,i)\in\mathcal{A}\}} x_{ji} = s_i \quad \forall i \in \mathcal{N}, \tag{39}$$

$$0 \leqq x_{ij} \quad \forall (i,j) \in \mathcal{A}, \tag{40}$$

where

$$s_1 = 1, \qquad s_t = -1,$$
$$s_i = 0 \quad \forall i \neq 1, t,$$

and t is the given destination.

The standard linear programming dual problem is

$$\begin{aligned} &\text{maximize} \quad p_1 - p_t \\ &\text{subject to} \quad p_i - p_j \leqq a_{ij} \quad \forall (i,j) \in \mathcal{A}, \end{aligned} \tag{41}$$

and by a classical duality theorem [Chv83], [Dan63], [PaS82], [Roc84], the optimal primal cost is equal to the optimal dual cost.

Let us associate with a given path $P = (1, i_1, i_2, \cdots, i_k)$ the flow

$$x_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are successive nodes in } P, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the CS conditions (1a) and (1b) are equivalent to the usual linear programming complementary slackness conditions

$$p_i \leqq a_{ij} + p_j \quad \forall (i,j) \in \mathcal{A},$$
$$0 < x_{ij} \Rightarrow p_i = a_{ij} + p_j \quad \forall (i,j) \in \mathcal{A}.$$

For a pair (x, p), the above conditions, together with primal feasibility (the conservation of flow constraint (39) for all $i \in \mathcal{N}$, which in our case translates to the terminal node of the path P being the destination node) are the necessary and sufficient conditions for x to be primal-optimal and p to be dual-optimal. Thus, upon termination of our shortest path algorithm, the price vector p is an optimal-dual solution.

To interpret the algorithm as a dual ascent method, note that a path contraction and an attendant price increase of the terminal node i of P, corresponds to a step along the price coordinate p_i that leaves the dual cost $p_1 - p_t$ unchanged if $i \neq 1$. Furthermore, an increase of the origin price p_1 by an increment δ improves the dual cost by δ. Thus the algorithm may be viewed as a finitely terminating dual coordinate ascent algorithm, except that true ascent steps occur only when the origin price increases; all other ascent steps are "degenerate," producing a price increase but no change in dual cost.

7. Computational results. The combined (forward and reverse) version of the algorithm without arc length scaling was implemented in a code called AUCTION_SP. This code solves the problem with a single origin and a selected set of destinations. It operates in cycles of iterations, alternating between the origin and one of the destinations. In particular, the algorithm first performs a group of (forward) iterations starting with the origin and proceeding up to the point where the origin again becomes the terminal node of the forward path; then the algorithm performs a group of (reverse) iterations starting at some destination, call it t, and proceeding up to the point where t becomes again the terminal node of the reverse path. The process is then repeated, starting again at the origin and then starting at another destination, and so on. The destinations are taken up cyclically, except that once the reverse path of some destination meets the forward path (in which case a shortest path for the given destination has been found), this destination is not iterated upon any further. Naturally, the same price vector p is used for the forward and all the reverse paths. The algorithm uses the default initialization ($p = 0$, $P = (1)$, $R = (t)$, for all destinations t), and terminates when each of the reverse paths have met the forward path.

We compared our code with the shortest path code SHEAP, due to Gallo and Pallotino [GaP88]. This is an implementation of Dijkstra's method that uses a binary heap to store the nodes which are not yet permanently labeled. We made a simple modification to this code so that it terminates when all the destinations (rather than all the nodes) become permanently labeled. Our informal comparison with other shortest path codes agrees with the conclusion of [GaP88] that SHEAP is a very efficient state-of-the-art code for a broad variety of types of shortest path problems. While other shortest path codes may produce faster solution times than SHEAP, we believe that the differences are not sufficiently large to invalidate the qualitative nature of our comparisons. We did not test our code against label correcting methods such as the threshold algorithm [GKP85], [GaP88], since these methods are at a disadvantage in the case of only a few origin-destination pairs.

We restricted our experiments to randomly generated shortest path problems obtained using the widely available NETGEN program [KNS74]. Problems were generated by specifying the number of nodes N, the number of arcs A, the length range $[1, L]$, and a single source and sink (automatically chosen by NETGEN to be nodes 1 and N). The times required by the two codes on a Macintosh II are shown in Tables 1 and 2, for the cases of one destination and four destinations, respectively. The tables show that AUCTION_SP is much faster than SHEAP on NETGEN problems; this was confirmed by extensive additional testing.

For the case of a single destination, we have also experimented with a version of SHEAP, called TWO_TREE_SHEAP, that builds a shortest path tree from the origin and another shortest path tree from the destination. Recent computational research [HKS88], [HKS89] has confirmed that using two trees in Dijkstra's method, as originally suggested in [Nic66], typically accelerates convergence, and our experience agrees with this conclusion. Still, however, AUCTION_SP was substantially faster than TWO_TREE_SHEAP, as shown in Table 1.

For multiple destination problems, we know of no Dijkstra-like algorithm that uses multiple trees; one has to run a two-sided algorithm separately for each origin-destination pair. Thus, in contrast with our algorithm, the advantage of a two-sided Dijkstra algorithm is dissipated quickly as the number of destinations increases from one. Therefore, based on our computational experience, we conclude that AUCTION_SP is by far the fastest code for random problems of the type generated by NETGEN and for few destinations (more than one, but much less than the maximum possible).

We note that for "one-to-all" problems, where there is a single origin and all other nodes are destinations, AUCTION_SP has been running slower than the best label correcting methods, including SHEAP. However, the differences in performance were not overwhelming (a factor of the order of two to three), and it will be interesting to make the corresponding comparison in a parallel computing environment.

The reader is warned that the computational results of the table are far from conclusive. Clearly, one can find problems where AUCTION_SP is vastly inferior to SHEAP in view of its inferior computational complexity, cf. Fig. 2 (although such a problem was never encountered in our experiments with randomly generated problems). An important issue is to delineate, through average complexity analysis and computational experimentation, the types of practical problems for which our algorithm is substantially better than the best label setting and label correcting methods. We find our computational results very encouraging, but further research and testing with both serial and parallel machines must be done before we can reach solid conclusions on the merits of our algorithm. We also note that the ideas in this paper are new and

TABLE 1

Solution times in secs of shortest path codes on a Mac II using problems generated by NETGEN *with one destination (node N). The lengths of all arcs were randomly generated from the range* [1, 1000].

N	A	AUCTION_SP	SHEAP	TWO_TREE_SHEAP
1,000	4,000	0.033	0.250	0.033
1,000	10,000	0.050	0.200	0.133
2,000	8,000	0.017	0.017	0.017
2,000	20,000	0.067	0.867	0.150
3,000	12,000	0.067	0.983	0.100
3,000	30,000	0.033	1.117	0.100
4,000	16,000	0.067	1.233	0.100
4,000	40,000	0.033	0.383	0.100
5,000	20,000	0.050	1.383	0.083
5,000	50,000	0.033	0.550	0.100

TABLE 2

Solution times in secs of shortest path codes on a Mac II using problems generated by NETGEN *with four destinations (nodes N, N − 100, N − 200, N − 300). The lengths of all arcs were randomly generated from the range* [1, 1000].

N	A	AUCTION_SP	SHEAP
1,000	4,000	0.050	0.250
1,000	10,000	0.080	0.383
2,000	8,000	0.100	0.667
2,000	20,000	0.233	0.883
3,000	12,000	0.117	1.100
3,000	30,000	0.167	1.117
4,000	16,000	0.100	1.233
4,000	40,000	0.117	1.883
5,000	20,000	0.150	1.533
5,000	50,000	0.183	1.833

their potential is not yet fully developed. It is likely that as these ideas are better understood, more efficient codes will become available.

REFERENCES

[AMO89] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Network flows*, Sloan Working Paper No. 2059-88, Sloan School of Management, Cambridge, MA, March 1989; also in Handbooks in Operations Research and Management Science, Vol. 1, Optimization, G. L. Nemhauser, A. H. G. Rinnooy-Kan, and M. J. Todd, eds., North-Holland, Amsterdam, 1989.

[Ber79] D. P. BERTSEKAS, *A distributed algorithm for the assignment problem*, Laboratory for Information and Decision Systems Working Paper, Massachusetts Institute of Technology, Cambridge, MA, March 1979.

[Ber81] ———, *A new algorithm for the assignment problem*, Math. Programming, 21 (1981), pp. 152-171.

[Ber82] ———, *Distributed dynamic programming*, IEEE Trans. Automat. Control, 27(1982), pp. 610-616.

[Ber86] ———, *Distributed relaxation methods for linear network flow problems*, in Proc. 25th IEEE Conference on Decision and Control, 1986, pp. 2101-2106.

[Ber88] ———, *The auction algorithm: A distributed relaxation method for the assignment problem*, Ann. Oper. Res. 14 (1988), pp. 105-123.

[Ber90] ———, *The auction algorithm for assignment and other network flow problems: A tutorial*, Interfaces, 20 (1990), pp. 133–149.

[BeE88] D. P. BERTSEKAS AND J. ECKSTEIN, *Dual coordinate step methods for linear network flow problems*, Math. Programming Ser. B, 42 (1988), pp. 203–243.

[BeT89] D. P. BERTSEKAS AND J. N. TSITSIKLIS, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[Chv83] V. CHVATAL, *Linear Programming*, W. H. Freeman, New York, 1983.

[Dan63] G. B. DANTZIG, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

[DGK79] R. DIAL, F. GLOVER, D. KARNEY, AND D. KLINGMAN, *A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees*, Networks, 9 (1979), pp. 215–248.

[Dia69] R. B. DIAL, *Algorithm* 360: *Shortest path forest with topological ordering*, Comm. ACM, 12 (1969), pp. 632–633.

[GaP86] G. GALLO AND S. PALLOTINO, *Shortest path methods: A unified approach*, Math. Programming Stud., 26 (1986), pp. 38–64.

[GaP88] G. GALLO AND S. PALLOTINO, *Shortest path algorithms*, Ann. Oper. Res., 7 (1988), pp. 3–79.

[GKP85] F. GLOVER, D. KLINGMAN, N. PHILLIPS, AND R. F. SCHNEIDER, *New polynomial shortest path algorithms and their computational attributes*, Management Science, 31 (1985), pp. 1106–1128.

[HKS88] R. V. HELGASON, J. L. KENNINGTON, AND B. D. STEWART, *Dijkstra's two-tree shortest path algorithm*, Tech. Report 89-CSE-32, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1988.

[HKS89] R. V. HELGASON, J. L. KENNINGTON, AND B. D. STEWART, *Computational comparison of sequential and parallel algorithms for the one-to-one shortest-path problem*, Tech. Report 89-CSE-32, Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, 1989.

[JoV87] R. JONKER AND A. VOLEGNANT, *A shortest augmenting path algorithm for dense and sparse linear assignment problems*, Computing, 38 (1987), pp. 325–340.

[KNS74] D. KLINGMAN, A. NAPIER, AND J. STUTZ, NETGEN—*A program for generating large scale (un) capacitated assignment, transportation, and minimum cost flow network problems*, Management Science, 20 (1974), pp. 814–822.

[Ker81] A. KERSHENBAUM, *A note on finding shortest path trees*, Networks, 11 (1981), pp. 399–400.

[Law76] E. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

[Nic66] T. NICHOLSON, *Finding the shortest route between two points in a network*, Comput. J., 9 (1966), pp. 275–280.

[Pap74] U. PAPE, *Implementation and efficiency of Moore-algorithms for the shortest path problem*, Math. Programming, 7 (1974), pp. 212–222.

[PaS82] C. H. PAPADIMITRIOU AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Pol91] L. POLYMENAKOS, *Analysis of parallel asynchronous schemes for the auction shortest path algorithm*, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1991.

[Roc84] R. T. ROCKAFELLAR, *Network flows and monotropic programming*, Wiley-Interscience, New York, 1984.

[ShW81] D. R. SHIER AND C. WITZGALL, *Properties of labeling methods for determining shortest path trees*, J. Res. Nat. Bur. Standards, 86 (1981), p. 317.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 448–474, November 1991

002

DIRECT SEARCH METHODS ON PARALLEL MACHINES *

J. E. DENNIS, JR.† AND VIRGINIA TORCZON†

Abstract. This paper describes an approach to constructing derivative-free algorithms for unconstrained optimization that are easy to implement on parallel machines. A special feature of this approach is the ease with which algorithms can be generated to take advantage of any number of processors and to adapt to any cost ratio of communication to function evaluation.

Numerical tests show speed-ups on two fronts. The cost of synchronization being minimal, the speed-up is almost linear with the addition of more processors, i.e., given a problem and a search strategy, the decrease in execution time is proportional to the number of processors added. Even more encouraging, however, is that different search strategies, devised to take advantage of additional (or more powerful) processors, may actually lead to dramatic improvements in the performance of the basic algorithm. Thus search strategies intended for many processors actually may generate algorithms that are better even when implemented sequentially. The key difference is that the additional processors are not used simply to enhance the performance of an inherently sequential algorithm; they are used to spur the design of ever more ambitious—and effective—search strategies.

The algorithms given here are supported by a strong convergence theorem, promising computational results on a variety of problems, and an intuitively appealing interpretation as multidirectional line search methods.

Key words. unconstrained optimization, direct search methods, multidirectional search, parallel optimization, Nelder–Mead simplex algorithm

AMS(MOS) subject classifications. 65K05, 49D30

1. Introduction.

We consider the nonlinear unconstrained optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}),$$

where $f : \mathbb{R}^n \to \mathbb{R}$. We do not use any derivatives or finite differences in our search schemes. These schemes qualify as *direct search methods* since the search is driven solely by function information. We require only that f be continuous on a compact level set to prove convergence for these methods; however, to guarantee convergence to a stationary point we require that f also be continuously differentiable. These convergence results will be discussed further in §4.

Originally, our interest in direct search methods was based on the fact that there had been only limited progress in designing effective parallel optimization algorithms that could take advantage of a large number of processors over a fairly wide range of problems. Meanwhile, current predictions suggest that achieving teraflop performance by the end of the century will require machines with 8000 to 32,000 processors. Workstations with up to 1000 processors are on the drawing board. What is needed are algorithms that can be easily *scaled* to accommodate ever larger numbers of processors—as well as ever more powerful processors. We believe that the algorithms we propose in this paper constitute progress in this direction.

The simplicity of direct search methods suggested to us that they might be easily adapted to a parallel computing environment precisely because they would be more amenable to scaling. A survey of the scientific literature also revealed that at least

* Received by the editors September 26, 1990; accepted for publication (in revised form) April 10, 1991. This research was sponsored by Air Force Office of Scientific Research grant AFOSR–89–0363. Equipment support was provided by the Center for Research on Parallel Computation, Rice University, Houston, Texas 77251-1892.

† Department of Mathematical Sciences, Rice University, Houston, Texas 77251–1892.

one direct search method, the Nelder–Mead simplex algorithm [16], numbers among the more popular optimization methods in scientific computing. The simplicity of the direct search methods certainly explains much of their popularity. It is also true that a lack of derivatives, as well as "noise" in the function values, may preclude the use of methods that require derivatives. Thus we believe that if we can use parallelism to improve the performance of direct search methods, any improvement will be of immediate interest and possible use. Recent experiments with problems from cancer research [1], chemical engineering [8], and stability analysis for matrix computations [9] have convinced us and our users that the approach given here is a valuable addition to the optimizer's toolkit.

The purpose of this paper is to describe these parallel direct search methods in the context of our earlier work and then to give some preliminary numerical results that indicate the potential for this approach. These results suggest the merit in pursuing the further development of parallel direct search methods. One goal is to demonstrate that, in the context of direct search methods, computing additional function values at each iteration can reduce the elapsed time to completion of an algorithm in a parallel computing environment and may actually reduce the *total* number of function evaluations required to produce an acceptable solution since the number of iterations required to reach a satisfactory solution may be significantly reduced. Thus, even though we are doing more work at each iteration (by computing more function values at each iteration), we learn so much more about the function that we produce significantly better iterates and thus converge in far fewer iterations. The net effect is that even with a more ambitious search strategy, we may actually compute fewer total function values.

Our paper is organized as follows. Section 2 outlines the basic approach we have taken to develop parallel direct search methods, and gives the assumptions we have made about the general parallel computing environment as well. Section 3 contains a brief description of direct search methods and the reason for our interest in them, as well as a comparison of our approach with several parallel implementations of quasi-Newton methods for solving the general unconstrained minimization problem. In §4 we review the multidirectional search algorithm, a direct search method that forms the basis for our more general parallel schemes. We also state the applicable convergence theorem from [22]. In §5, we show how to incorporate the basic multidirectional search algorithm into a core step that can then be augmented to take better advantage of the available computational resources while retaining the convergence properties of the basic algorithm. In §6 we outline the new parallel multidirectional search algorithms and discuss implementation details. In §7, we give some preliminary numerical results that demonstrate speed-up on two fronts and report our experience with some "real" problems. In §8, we close with some remarks concerning future directions for research.

2. Approach. The approach we take can be viewed as an extremely flexible multidirectional line search method that can be easily scaled to fit the number of processors available—regardless of the size of the problem to be solved. Complete use of the available processors is accomplished in two ways. First, additional search directions are introduced systematically in an order that in some sense reflects the likelihood of producing descent. In addition, the simple line search conducted along each direction is refined, with preference given to those directions that are deemed more likely to produce descent. The work involved in determining the search directions and steps is minimal; we do not need to solve any linear systems of equations. Furthermore, this approach involves a minimum of interprocessor communication (i.e.,

synchronization) and places few restrictions on the function to be minimized. In particular, there is no need to assume that the function evaluations are expensive in order to justify the overhead introduced by the parallelization. This added flexibility is significant. If the function evaluations must be expensive to support the cost of the synchronization, then as processors become ever faster, the range of problems that can be solved efficiently on parallel machines becomes ever smaller. This limitation will become even more of an issue as the number of processors grows since the cost of access to remote memory will become even greater. As we shall see, one of the most attractive features of the parallel direct search methods is that these methods allow us to stack function evaluations on each processor until the cost of the computation balances the cost of the communication.

Throughout this paper we will assume that $n < p$, where n is the dimension of the problem to be solved and p is the number of available processors. While it is certainly possible to use these algorithms when $p \leq n$, the more interesting results occur when, in fact, $n \ll p$ (or, more accurately, when the total number of function values computed at each iteration of the algorithm is significantly larger than the dimension of the problem to be solved).

The results we will report in §7 have been taken from an implementation on an iPSC/860, but these algorithms can be adapted to any sort of parallel computing environment. This certainly includes either distributed-memory or shared-memory multiprocessors. However, since these algorithms are both small and flexible, they also are amenable for use on transputers or even on a network of computers that may or may not have different performance characteristics. There is only one point of synchronization, so modifications to suit a particular parallel computing environment are straightforward. Furthermore, the information we require for the synchronization is so small, regardless of the search strategy we employ, that even when global communication or some other form of access to remote memory is relatively expensive, it is still easy to choose a search strategy from among those we propose for which this approach is viable. We have concentrated on multiple instruction, multiple data (MIMD) machines only because we are ultimately interested in solving problems where the function values are themselves the result of another (expensive) process, for instance, a simulation or the solution of a differential equation. We choose to treat the function evaluation routine as a "black box." However, with the proper restrictions on the function evaluation routine, the ideas presented here could also be implemented on single instruction, multiple data (SIMD) machines. Thus we have an approach that is flexible enough to be of use in a wide range of computing environments.

3. Background. The methods given here belong to the large and often-used class called direct search methods. Direct search methods are characterized by the fact that they do not use derivatives. Derivative-free schemes are more widely applicable than gradient or quasi-Newton methods. Of course, there is little doubt that derivatives, when they are available, can be used to speed up the average-case performance of nonlinear optimization algorithms, but sometimes derivative approximations are simply not practical. In some control problems the objective calculation is so expensive that finite differences are undesirable and the code involves so much branching that automatic differentiation is currently out of the question, while an expert might require weeks, or even years, to find an alternative adjoint approach to derivative approximation. Derivative-free methods are also quite robust in dealing with objective functions that can be evaluated to only a few significant digits, which precludes the use of finite-difference derivative approximations, regardless of expense.

As an indication of how popular direct search methods are with users, one need only consult the 1989 Science Citation Index [18], which lists more than 215 citations for the classic Nelder–Mead paper. Both the number of citations and the range of journals in which these citations occur has grown every year since the paper first appeared in 1965. The most cited paper in the vast literature on quasi-Newton methods appears to be the 1963 paper of Fletcher and Powell [7], which popularized what is now known as the DFP variable metric secant update. In the 1989 Science Citation Index the Fletcher–Powell paper has 114 citations. As a further indication of its popularity, we note that the Nelder–Mead simplex algorithm also appears in most commercially available software libraries. For instance, Nelder–Mead is a standard feature of such packages as NAG, IMSL, and Matlab.

Given the popularity of the Nelder–Mead simplex algorithm, our first attempt at a parallel direct search method consisted of a straightforward implementation of the Nelder–Mead simplex algorithm on an iPSC hypercube (now referred to as an iPSC/1). While we computed $n + 4$ function values simultaneously to complete all the function evaluations that could possibly be required during the course of a single iteration, the algorithm showed only a speed-up of order two, regardless of either the size of the problem or the number of available processors. Careful examination of the behavior of the *sequential* algorithm confirmed that, in fact, this was the best we could expect to do with such a naive parallel implementation since the sequential algorithm typically required only two function values per iteration. Additional experimentation led to two interesting results. The first was the development of a new direct search method, which we call multidirectional search [21], that forms the basis for the algorithms discussed here. The second was the unexpected discovery, during our numerical testing, that the Nelder–Mead simplex algorithm can converge to nonminimizers when the dimension of the problem becomes large enough [21]. This behavior occurred for all the test problems we used from the Moré, Garbow, Hillstrom problem set [12] where the dimension of the problem could be varied. In all cases, we started with a regular simplex (i.e., a simplex with edges of equal length) from the standard starting point given in [12]. This behavior occurred even on such a simple problem as

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) = \mathbf{x}^T\mathbf{x}$$

for $n \geq 16$. One of the advantages of the multidirectional search algorithm is that, unlike the Nelder–Mead simplex algorithm, it is backed by convergence theorems that, our numerical testing indicates, are borne out in practice.

Initially our implementation of the basic multidirectional search algorithm on a shared-memory multiprocessor suffered from the fact that the number of processors that could be used was tied to the dimension of the problem. The challenge, then, was to find ways to effectively use all available processors.

Our approach consists of embedding the original multidirectional search algorithm in a family of algorithms. These algorithms have a very appealing interpretation as multidirectional line search methods. The original multidirectional search algorithm [21] performs a rudimentary line search along n search directions, hence the dependence of the use of processors on the dimension of the problem. Our new approach expands upon this idea by systematically introducing line searches along new search directions while further refining the line search along the existing set of search directions. As we shall see, this approach is extremely flexible, so that even for a problem of a given dimension on a machine with a fixed number of processors, there

is a family of closely related algorithms, any one of which could be implemented.

The idea for using all available processors, or *scaling* the algorithm to fit the properties of a given machine, is an idea that we have seen in other parallel optimization algorithms. The key difference is that we do not use additional processors simply to enhance the performance of an inherently sequential algorithm.

A Newton or quasi-Newton method is essentially a sequential algorithm that consists of two distinct phases. First, a *single* search direction is constructed and then a step that satisfies some notion of sufficient decrease in the objective function value is determined. These methods are inherently sequential since a step for the current iteration cannot be determined until after the search direction has been constructed, while the search direction for the next iteration cannot be ascertained until a successful step has been found.[1] There can, however, be a significant amount of work associated with each phase of the iteration. Thus, efforts to produce general, parallel Newton or quasi-Newton methods have concentrated on parallelizing the work involved in each phase [6], [3], [4], [13], [14], [15]. These efforts have been successful at accelerating the performance of Newton or quasi-Newton methods while preserving their convergence properties. However, the inherently sequential nature of Newton's method limits the number of processors that can be successfully employed since the fundamental algorithm remains essentially unchanged. These limitations arise in two ways. Either a fairly small number of processors (i.e., no more than twenty) can be used to parallelize the linear algebra involved at each iteration (but this requires the assumption that the dimension of the problem is quite large so as to offset the cost of the synchronization), or the processors can be used to calculate finite-difference approximations to the gradient and either part or all of the Hessian. Then, for a fixed number of processors, the range of the problems that can be solved is limited, both by the dimension of the problem (i.e., $O(n) < p < O(n^2)$, where p is the number of processors and n is the dimension of the problem) and by the relative cost of the function evaluations, again to offset the cost of the synchronization.

Our approach is quite different. Given the dimension of the problem to be solved, the number of available processors, and, ideally, some notion of the relative expense of the function evaluations to communication (or synchronization) costs, we have a simple initialization scheme that tailors the basic multidirectional search algorithm to fit these specifications. The result is not just that we produce a different sequence of iterates. We actually produce *different* algorithms with *different* performance characteristics. The numerical results in §7 suggest that we often generate better direct search algorithms. This improved performance is not accidental. We compute more information about the function—more than we could easily justify in a sequential computing environment—but we use *all* of the information we compute. Not too surprisingly, we often construct a better sequence of iterates.

Before proceeding to a description of the multidirectional search algorithm, we note that this is not the only direct search method we could use as a core step for our more general parallel direct search schemes. Our investigation of the theoretical properties of the multidirectional search algorithm revealed that several well-established sequential direct search methods, such as the original factorial design algorithm of

[1] Byrd, Schnabel, and Shultz [3], [4] *anticipate* the search direction for the next iteration by calculating, speculatively, the gradient associated with the trial step. The calculation is speculative in the sense that the outcome of the trial step determines whether or not their guess was correct and thus whether or not the information they have already computed can be used to calculate the new search direction.

Box [2] or the pattern search algorithm of Hooke and Jeeves [10], also have the same essential characteristics, and thus are amenable to both the convergence analysis given in [22] and the general parallelization strategy given here in §5. The reason we have chosen to use the multidirectional search algorithm as our basic algorithm is that it requires only $O(n)$ function evaluations per iteration to guarantee convergence. The factorial design algorithm also falls under the same convergence analysis but requires $O(n^2)$ function evaluations per iteration. Thus, when there is a limited number of processors available (relative to the size of the problem), the multidirectional search algorithm is more effective. Conversely, when the number of processors is quite large, it is easy to construct examples for which the multidirectional search algorithm and the factorial design algorithm of Box generate the same generalized direct search algorithm. The similarities between these and other direct search algorithms are discussed briefly in [22]; a more detailed discussion is being prepared for publication.

We now proceed with a description of our core algorithm.

4. The basic multidirectional search algorithm. An iteration of the basic multidirectional search algorithm begins with a simplex S in $\mathbb{R}^n$, with vertices $\mathbf{v}_0, \mathbf{v}_1, \cdots \mathbf{v}_n$. The best vertex $\mathbf{v}_0$ is designated to be a vertex for which $f(\mathbf{v}_0) \leq f(\mathbf{v}_j)$ for $j = 1, \cdots, n$. We now describe a complete iteration to arrive at a new simplex S_+.

The first move of the iteration is to reflect $\mathbf{v}_1, \cdots, \mathbf{v}_n$ through the best vertex $\mathbf{v}_0$. Figure 1 shows an example for $n = 2$. The reflected vertices are labeled $\mathbf{r}_1$ and $\mathbf{r}_2$.

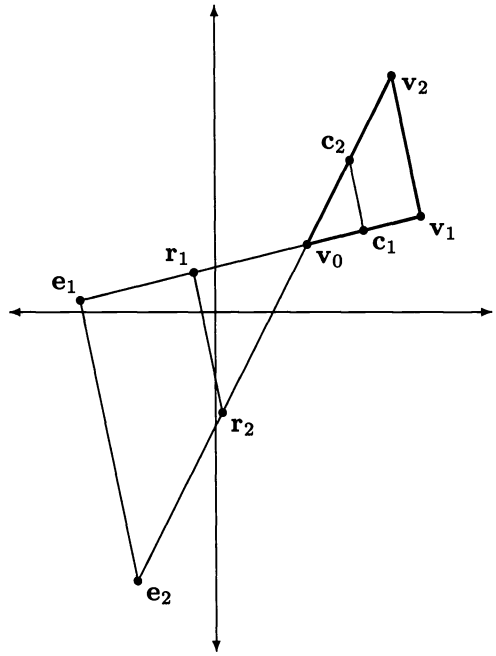


FIG. 1. *The three possible steps given the simplex S with vertices $\langle \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2 \rangle$.*

If a reflected vertex gives a better function value than the best vertex, then the *reflection* step is called successful and the algorithm tries an *expansion* step. The expansion step consists of expanding each reflected edge $(\mathbf{r}_j - \mathbf{v}_0)$ to twice its length to give a new expansion vertex $\mathbf{e}_j$. In Fig. 1 the expansion vertices are labeled $\mathbf{e}_1$ and $\mathbf{e}_2$.

In an iteration of this basic algorithm, the expansion step would be tried only if the reflection step was successful, and it would be taken only if some expansion vertex was better than all the reflection vertices. Thus, if we try the expansion step, then the new simplex S_+ is either the expansion simplex ($\langle \mathbf{v}_0, \mathbf{e}_1, \mathbf{e}_2 \rangle$ in Fig. 1) or the reflection simplex ($\langle \mathbf{v}_0, \mathbf{r}_1, \mathbf{r}_2 \rangle$ in Fig. 1).

The other branch of the basic algorithm is the case where the reflection step was unsuccessful, i.e., no reflection vertex has a better function value than $f(\mathbf{v}_0)$. In this case, we take S_+ to be the contraction simplex formed by replacing each vertex of the worst n-face in the original simplex by the point midway from it to the best vertex. Thus, in Fig. 1, the *contraction* step takes S_+ to be $\langle \mathbf{v}_0, \mathbf{c}_1, \mathbf{c}_2 \rangle$.

To complete one iteration of the basic algorithm, we take $\mathbf{v}_0^+$ to be the best vertex of S_+.

Before giving the convergence result, we point out the line search flavor of the algorithm. In the case $n = 1$, we first try a step of a given length away from the vertex with the larger function value (the reflection step). If that is successful, then we try a longer step (the expansion step), but if it is not, then we try a step only half as long in the other orientation (the contraction step). If none of the steps give decrease, then the next iteration begins with a step in the same direction as the previous iteration began with, but only half as long. Thus, an unsuccessful sequence of iterations generates a backtracking line search with alternating orientations. (See Fig. 2.) This simple observation forms an important part of the convergence proof.

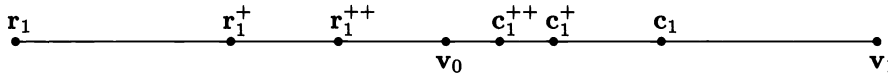


FIG. 2. *A simple backtracking line search.*

The following notation will be used in the statement of the convergence result. Let $\{\mathbf{v}_0^k\}$ be the sequence of best vertices. Let $\mathbf{v}_0^0$ be the best vertex of the simplex S_0 used to start the algorithm. Define the level set of f at $\mathbf{v}_0^0$ to be

$$L(\mathbf{v}_0^0) \quad = \quad \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{v}_0^0)\}.$$

Given $\mathbf{y} \in \mathbb{R}^n$, let the contour $C(\mathbf{y})$ be

$$C(\mathbf{y}) \quad = \quad \{\mathbf{x} : f(\mathbf{x}) = f(\mathbf{y})\}.$$

Let X_* be the set of stationary points of the function f in $L(\mathbf{v}_0^0)$.

THEOREM 4.1. *Assume that $L(\mathbf{v}_0^0)$ is compact and that f is continuously differentiable on $L(\mathbf{v}_0^0)$. Then some subsequence of $\{\mathbf{v}_0^k\}$ converges to a point $\mathbf{x}_* \in X_*$. Thus, $\{\mathbf{v}_0^k\}$ converges to $C_* = C(\mathbf{x}_*)$ in the sense that*

$$\lim_{k \to \infty} \left[\inf_{\mathbf{x} \in C_*} \|\mathbf{v}_0^k - \mathbf{x}\| \right] \quad = \quad 0.$$

The assumption that f is continuously differentiable on $L(\mathbf{v}_0^0)$ can be reduced to the assumption that f is continuous on $L(\mathbf{v}_0^0)$; however, the set X_* must then be expanded to include all points where the function f is nondifferentiable on $L(\mathbf{v}_0^0)$ and where the gradient of f exists but is not continuous. The proof for both results is given in [22]. Before we go on to extend the multidirectional search algorithm to

generate a family of parallel algorithms, let us discuss the proof in a form that extends to the algorithms of the next section.

First, we see why the algorithm cannot stall at a $\mathbf{v}_0$ with a nonzero gradient. Note that the edges of S adjacent to $\mathbf{v}_0$ form a basis for $\mathbb{R}^n$ and so at least one edge is not orthogonal to $\nabla f(\mathbf{v}_0)$. Thus, either that edge or its reflection is a descent direction from $\mathbf{v}_0$. Let us call this a descent edge. Now, the only way the algorithm can stay at $\mathbf{v}_0$ is to take an infinite sequence of successive contraction steps. However, at the next iteration, the contraction simplex flips to the opposite orientation to form the reflection simplex and so, along any descent edge from $\mathbf{v}_0$, we are generating a pair of sequences of points, one from each orientation, halving the distance from $\mathbf{v}_0$ at each term of the sequence, as seen in Fig. 2. Therefore, we will eventually get either a successful reflection step or a contraction step that replaces $\mathbf{v}_0$.

Thus, the algorithm can be viewed as a backtracking line search method where at least one of the n search directions is guaranteed to produce descent if $\nabla f(\mathbf{v}_0) \neq 0$. Convergence would be clear if some principle of sufficient decrease on f were required. However, we accept a new best vertex based only on simple decrease. The remainder of the proof consists of an unusual argument by contradiction. We assume that the sequence of best vertices stays uniformly bounded away from the set of stationary points. Using this assumption, along with the compactness of the level sets, the uniform linear independence of the search directions, and the continuity of ∇f, we can show that all but a finite number of vertices generated by the algorithm must be contained in a compact set and lie on a lattice. Now, the first part of the proof showed that if the best vertex is not a stationary point of the function, then the algorithm will produce a *strictly* monotonically decreasing sequence of function values. The second part of the proof demonstrates that under the hypothesis that the sequence of best vertices stays uniformly bounded away from the the set of stationary points there is only a *finite* number of function values. Therein lies the contradiction. Thus, our hypothesis cannot hold and we have convergence to the set of stationary points.

We close by noting that as long as we preserve the backtracking line search flavor of the algorithm, the proof for the basic algorithm will extend to the parallel multidirectional search algorithms we propose.

5. The parallel multidirectional search algorithms. The strategy we will employ to define a family of direct search algorithms is very simple. We will look ahead to subsequent iterations of the algorithm until we generate a sufficient number of vertices to keep all available processors busy.

We begin by removing all the branching from the basic algorithm to obtain a *core* step. Thus, the core step consists of the union of the reflection, expansion, and contraction steps from the basic algorithm. This core step will require $3n$ independent function values at each iteration.[2] Thus, in the two-dimensional example given in Fig. 1, the core algorithm computes the function values at the six new vertices $\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{c}_1$, and $\mathbf{c}_2$ simultaneously. We then choose as $\mathbf{v}_0^+$ the vertex that produces the best function value while S_+ is taken to be the simplex that produced $\mathbf{v}_0^+$. If $f(\mathbf{v}_0)$ is still the least function value, then S_+ must be the contraction simplex.

There are two points to be made. First, with the stipulation that the contraction simplex must be accepted when the core step does not produce a new best vertex, the convergence theorem still holds. Second, without the branching present in the basic

[2] We will assume for now that the number of processors, p, is greater than $3n$. As we shall demonstrate in §6.3, it is possible to remove this restriction on the minimum number of processors required. In fact, as we shall see in §7, this approach may generate better *sequential* algorithms.

algorithm, we may actually produce a different sequence of iterates. For example, our choice of S_+ might now be the expansion simplex even if it would never have been constructed in the basic algorithm. This could happen if $\min\{f(\mathbf{e}_1), \cdots, f(\mathbf{e}_n)\} < f(\mathbf{v}_0) \leq \min\{f(\mathbf{r}_1), \cdots, f(\mathbf{r}_n)\}$. Thus we have a different algorithm that may actually produce a different sequence of iterates.

We have now used $3n$ processors to compute the $3n$ new vertices and their associated function values. To take advantage of more processors, we simply continue the look-ahead to subsequent iterations. For instance, we could assume that the reflection step is accepted at the current iteration, in which case one of the n new vertices associated with the reflection simplex will become $\mathbf{v}_0^+$. Thus, we can consider the new reflection vertices that might be constructed at the next iteration if each of $\mathbf{r}_1, \cdots, \mathbf{r}_n$ were given the role of $\mathbf{v}_0^+$. (See Fig. 3.) We can continue this look-ahead to construct all the reflection simplices that could be considered at the next iteration if any of $\mathbf{r}_1$, $\cdots$, $\mathbf{r}_n$, $\mathbf{e}_1$, $\cdots$, $\mathbf{e}_n$, $\mathbf{c}_1$, $\cdots$, $\mathbf{c}_n$, or $\mathbf{v}_0$ were to become $\mathbf{v}_0^+$, as shown in Fig. 4. There is also nothing to prevent us from including all possible expansion and contraction vertices as well, as seen in Fig. 5.

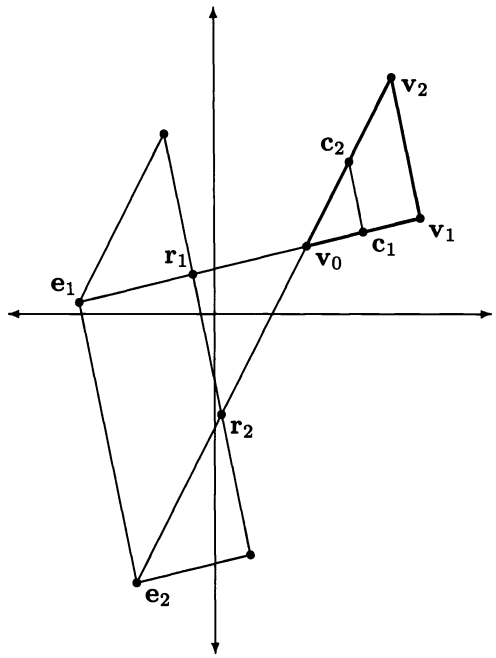


FIG. 3. *The core step with reflection steps from* $\mathbf{r}_1$ *and* $\mathbf{r}_2$.

If we were to construct all the new vertices shown in Fig. 5 and compute their associated function values in parallel, then we would have effectively completed two iterations of the basic multidirectional search algorithm. However, the numerical results we will show in §7 suggest that, in fact, we typically do much better, for reasons we will now explain.

If we return to the core step, we see that all the vertices we constructed lie along n directions determined by the n edges adjacent to $\mathbf{v}_0$, as seen in Fig. 6. We can view the core step as a rudimentary line search consisting of three steps along each of n directions. When we proceed to the next iteration and consider all possible reflection simplices, we introduce new line searches but we also further refine the search along the

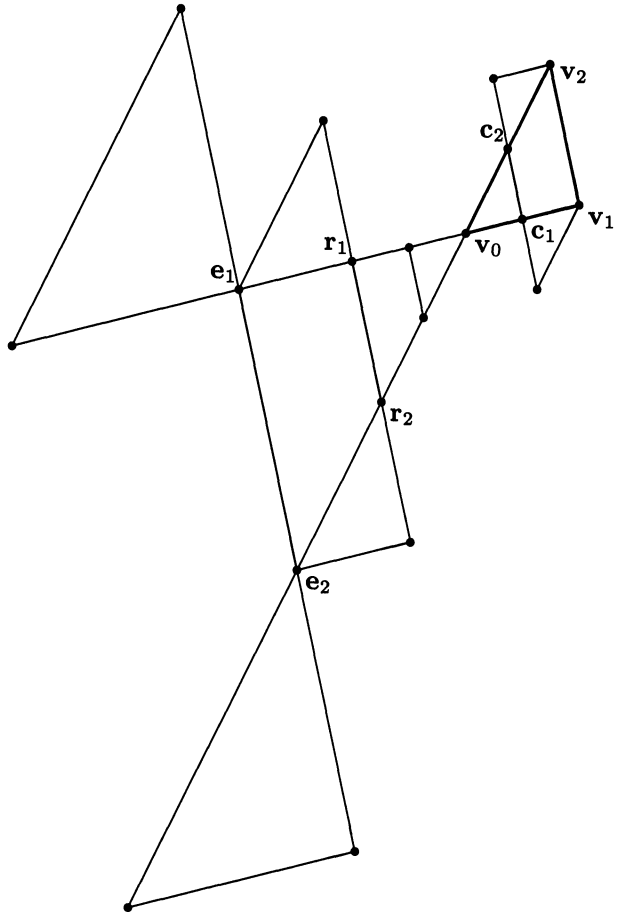


FIG. 4. *The core step with all the reflection steps for the next iteration.*

original n directions, as can be seen in Fig. 7. When we complete the full look-ahead, Fig. 8 shows that for the example we have constructed we have introduced five new line searches, each consisting of three steps, while simultaneously adding additional steps along the two original search directions to refine the line search. If we were to continue this process into the next iteration we would find that we would again introduce new line searches, we would begin to refine the search along the directions introduced in the previous iteration, and we would further refine the searches along the original n directions.

Figure 8 suggests that we have a true multidirectional line search method that scales the algorithm by introducing new line searches in a systematic way. The original n search directions are deemed most likely to produce descent since we search along edges from vertices with higher function values towards a vertex with a lower function value. In fact, the convergence theorem guarantees that if $\mathbf{v}_0$ is not a stationary point, then one of these n edges will produce descent. However, we hedge our bets by also adding new, less likely, search directions. Throughout this process we continue to refine the line search along each of the directions we have introduced, with priority going to the more likely directions.

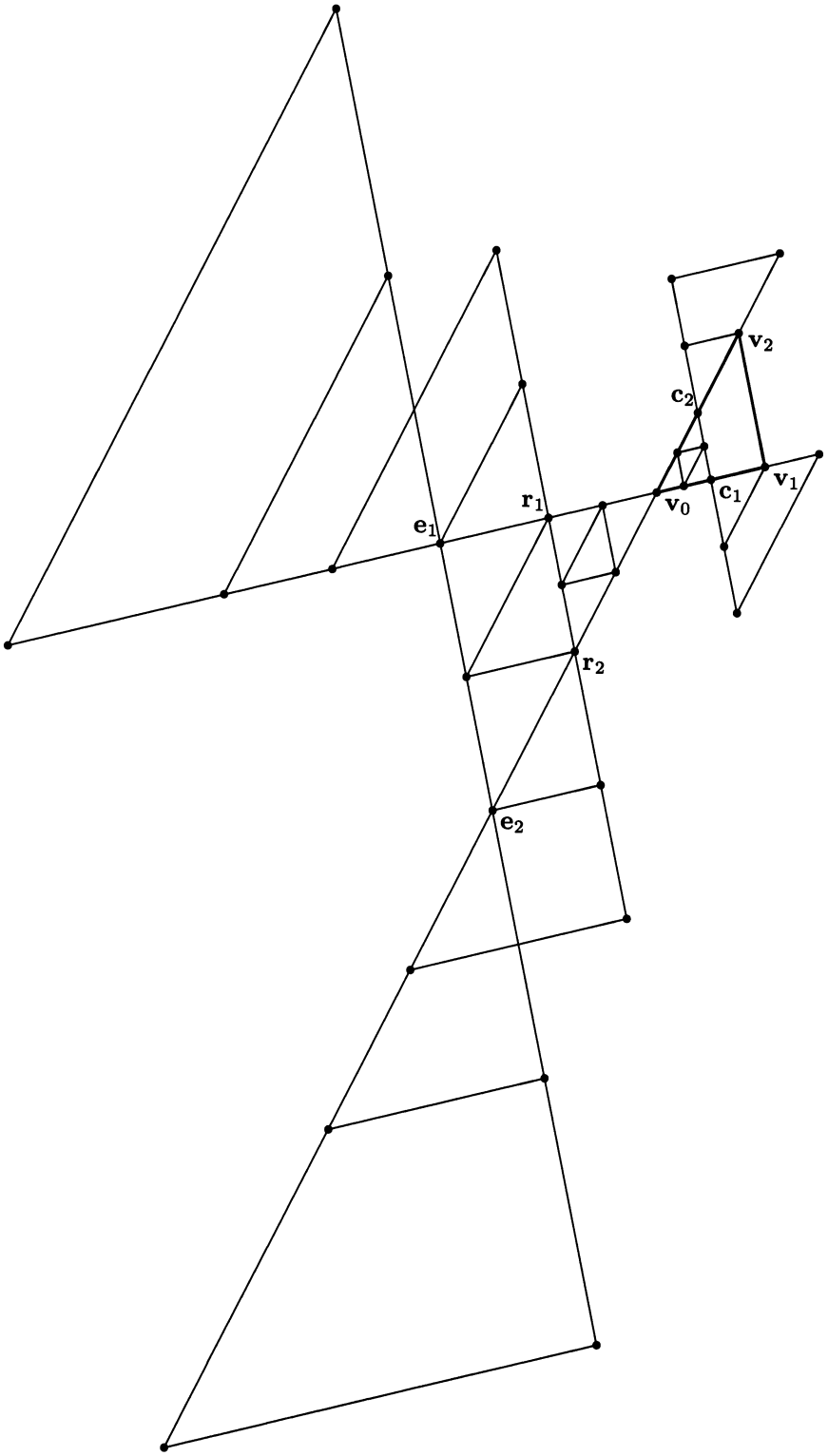


FIG. 5. *The core step with one complete look-ahead.*

We prefer to interpret our direct search methods as multidirectional line search algorithms. The simplex interpretation is useful for generating and programming these algorithms, as we shall see in the next section; however, the line search interpretation

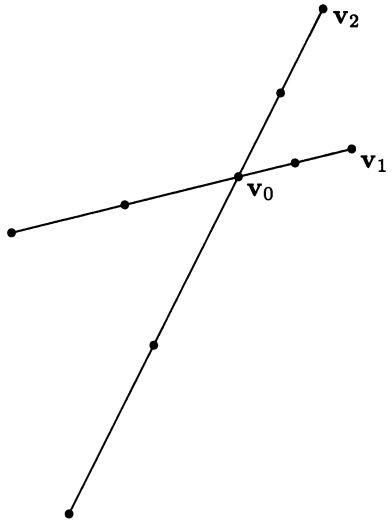


FIG. 6. *The n original search directions.*

allows us to pose these algorithms as gradient-related methods, which helps explain both the convergence theory and the performance characteristics of these algorithms.

We also note that we have introduced a *family* of algorithms, not a single method. We have specified a scheme for introducing search directions and refining line searches, but there is tremendous flexibility within this scheme. In the example we have shown in Fig. 8, we have introduced 33 new vertices. On a 32 processor machine we can choose almost any subset consisting of 32 of these 33 vertices to define a multidirectional search algorithm. While each of these subsets was generated using the same look-ahead scheme, each subset produces a distinct algorithm that may generate a different sequence of iterates when applied to identical problems. This observation suggests the need for defining strategies to specify the order in which to introduce search directions and refine line searches. The only limitation we impose arises from the convergence theorem: we must ensure that the backtracking line search, which constructs steps along *both* orientations of the search edge, is preserved.

Defining a strategy is not difficult. We used the following principles to design our current implementation of the parallel multidirectional search algorithms:

- We construct a list of vertices until we have enough vertices to assign to all the processors.
- We start this list with $\mathbf{v}_0$ as the seed. We consider each vertex in the order in which it was added to the list and generate the *complete* core step associated with that vertex. The $3n$ vertices associated with a complete core step are then added to the bottom of the list of vertices.
- We give precedence to the reflection step, then the contraction step, and then finally the expansion step when adding vertices to the list.
- We include the current best vertex, with reduced edge lengths, only as part of the contraction step, since in the basic algorithm there would be a new

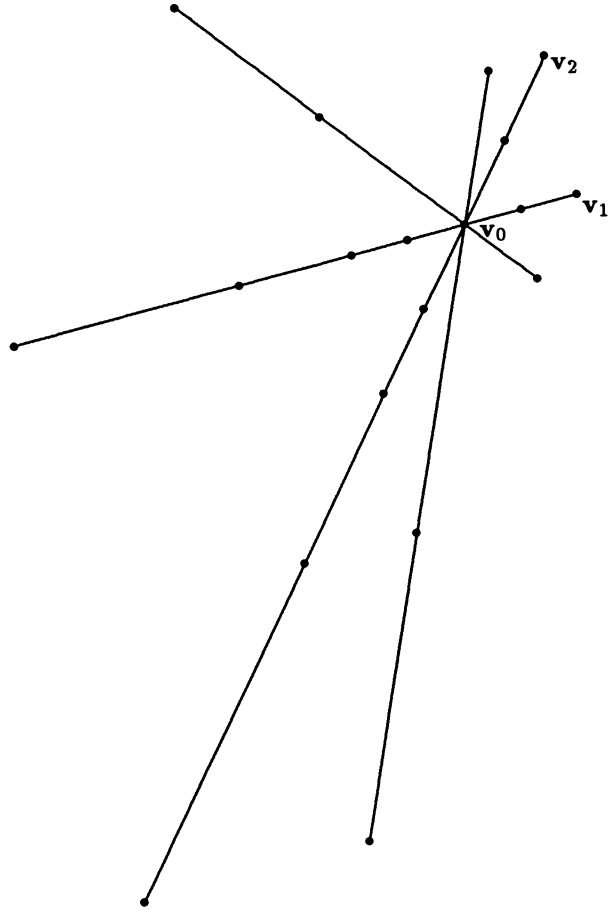


FIG. 7. *The n original searches with* new *steps and* additional *search directions.*

best vertex if we accepted either the reflection or expansion steps.
The actual algorithm we use to implement this strategy is discussed in more detail in the next section.

There are certainly other, possibly better, strategies for generating parallel multidirectional search algorithms. For instance, we could first construct all the reflection vertices associated with a single iteration as in Fig. 4 and then all the contraction vertices, etc. We could also have mixed strategies that allow for different choices depending on the type of step that produced decrease in the previous iteration. These ideas are the subject of future research and will be discussed further in §8.

Another important point to note is that once we have specified a strategy for generating both the search directions and the steps, every vertex can be represented as a *fixed* linear combination of $\mathbf{v}_0$ and the edges adjacent to $\mathbf{v}_0$. There is no need to regenerate the necessary coefficients at every iteration. To see this, consider our example in Fig. 7. Since $(\mathbf{v}_1 - \mathbf{v}_0)$ and $(\mathbf{v}_2 - \mathbf{v}_0)$ span $\mathbb{R}^2$, each of the new vertices can be defined as the sum of $\mathbf{v}_0$ and a linear combination of $(\mathbf{v}_1 - \mathbf{v}_0)$ and $(\mathbf{v}_2 - \mathbf{v}_0)$. If we fix these coefficients, and then vary S, computing the new vertices at each iteration of the search reduces to computing a linear combination of the edges adjacent to the new best vertex. Thus we have a *template* for the search that is defined by our choice

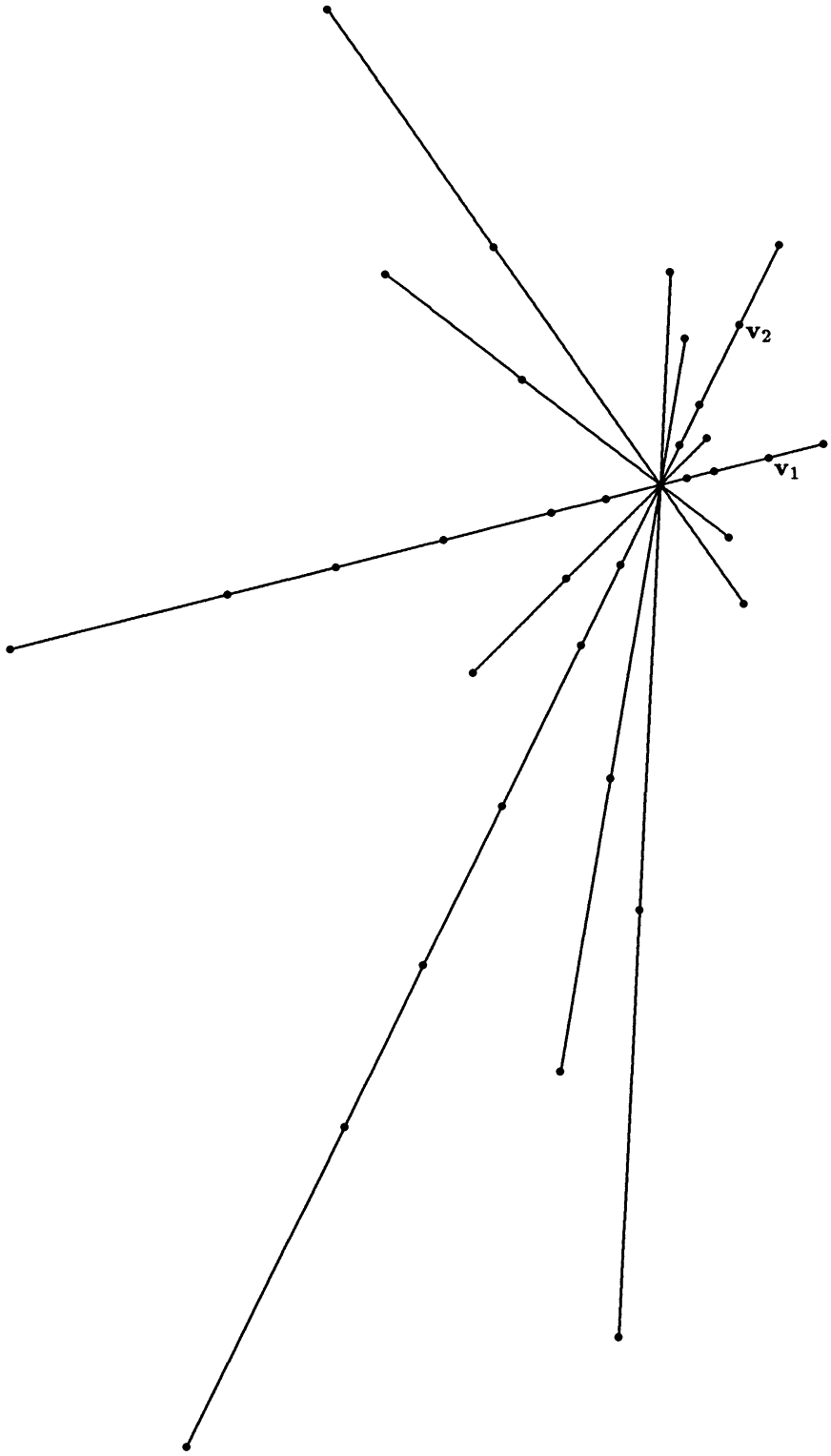


FIG. 8. *A different interpretation of the core step with one complete look-ahead.*

of strategies before the actual search procedure begins. Again, we will defer further discussion of this point to the next section.

Another advantage of the static initialization scheme is that it allows us to eliminate duplicate vertices in the template. The reader has probably already noticed that once we begin to look ahead to subsequent iterations, some vertices may be multiply-defined. For example, in Fig. 3, $\mathbf{e}_1$ and $\mathbf{e}_2$ are redefined by the new reflection simplices. However, the coefficients necessary to construct these vertices from $(\mathbf{v}_1 - \mathbf{v}_0)$ and $(\mathbf{v}_2 - \mathbf{v}_0)$ are identical, so such duplication is easy to detect and eliminate during the initialization. The only other issue to be decided is which simplex will be associated with a multiply-defined vertex, information that is needed to avoid ambiguity when defining S_+ in the event that this vertex becomes $\mathbf{v}_0^+$. We resolve this issue by breaking ties in favor of the first simplex to define the vertex.

Having anticipated some of the major points to be addressed in any implementation of the parallel multidirectional search algorithms, we are now ready to turn to a more detailed discussion of our current implementation.

6. A distributed memory implementation. We begin with a statement of the basic algorithm, shown in Table 1. Each of the p processors[3] constructs one vertex $\mathbf{v}_i$ and its function value fv_i. The scalars $a_i, \cdots, z_i$ required to construct the vertex are local to the processor. We assume that the bulk of the computation occurs in the evaluation of $f(\mathbf{v}_i)$. Note that we have a single program running on each of the processors. The data, in the form of the scalars required to compute the vertex $\mathbf{v}_i$, varies on each processor. Thus we have a single program, multiple data (SPMD) model. With the appropriate restrictions on the function evaluation routine, which we choose to treat as a "black box," these methods could also be extended to SIMD machines.

TABLE 1
The parallel multidirectional search algorithm.

Given an initial simplex S_0 with vertices $\langle \mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_n \rangle$,
  initialize template
  while (stopping criterion is not satisfied) **do**
    for $i = 1, \cdots, p$ **do**
      $\mathbf{v}_i \leftarrow \mathbf{v}_0 + a_i(\mathbf{v}_1 - \mathbf{v}_0) + \cdots + z_i(\mathbf{v}_n - \mathbf{v}_0)$
      $fv_i \leftarrow f(\mathbf{v}_i)$
    end
    $fv_* \leftarrow \min_i \{fv_i\}$ /* communication */
    update simplex
  end

When each of the processors has completed its function evaluation, there is a single global exchange to determine the least function value. On the iPSC/860 we can exploit the hypercube connectivity by using a global handshake algorithm in which each processor exchanges the least function value it has seen with its nearest neighbor. If we assume a hypercube of dimension d, once each processor has exchanged information

[3] We assume, for now, enough processors to compute the $3n$ vertices associated with a core step. As we shall see in §6.3, satisfying this requirement—even when we are working on a single processor—is straightforward.

with its d nearest neighbors, every processor has fv_*. Note that this eliminates the need for a single controlling process since each processor can also test for convergence. To adapt this algorithm to a different parallel computing environment, one need only make the appropriate modifications at this single point of synchronization to introduce the appropriate form of remote memory access.

6.1. Update. During the course of the global exchange, we pass three additional pieces of information: the vertex $\mathbf{v}_*$ that produced fv_*, the pointer *source* that corresponds to the vertex in S that produced $\mathbf{v}_*$, and a scalar α_* that, in conjunction with $\mathbf{v}_*$ and *source*, allows us to construct the simplex S_* associated with $\mathbf{v}_*$. (The pointer *source* and the scalar α_i associated with the vertex $\mathbf{v}_i$ are local to each processor and are assigned during the initialization of the template.) With this additional information, updating S to produce S_+ is straightforward, as seen in Table 2. Note that before we update S, we check to see if fv_* produced the strict decrease we require. If not, then $\mathbf{v}_0$ is still the best vertex; to ensure convergence, we reduce the lengths of the edges in the simplex S by the contraction factor $\theta \in (0,1)$.

TABLE 2
Updating the simplex.

if $(fv_* < fv_0)$ **then**
  $\mathbf{v}_0^+ \leftarrow \mathbf{v}_*$
  $\alpha_+ \leftarrow \alpha_*$
else
  $\mathbf{v}_0^+ \leftarrow \mathbf{v}_0$
  $\alpha_+ \leftarrow \theta$
  $source \leftarrow 0$
endif
for $j = 0, \cdots, n$ **do**
  $\mathbf{v}_j^+ \leftarrow \mathbf{v}_0^+ + \alpha_+(\mathbf{v}_j - \mathbf{v}_{source})$
end
swap(0,source)

6.2. Initialization. The real effort in defining a parallel multidirectional search algorithm lies in the initialization. The key point to emphasize is that this initialization is *static*. A strategy is defined before the search actually begins. The strategy can be specified as a template that is *fixed*; it is the simplex, and not the template, that varies from iteration to iteration. The template is possible because each vertex in the search scheme can be represented as a sum of $\mathbf{v}_0$ and a unique linear combination of the edges adjacent to $\mathbf{v}_0$, as demonstrated in Fig. 9. Furthermore, once we have $\mathbf{v}_0^+$, it is possible to construct S_+ from S with just two additional pieces of information, α_+ and *source*, as shown in Fig. 10.

To generate the coefficients $a_i, \cdots, z_i$ we need to construct the vertex $\mathbf{v}_i$, and the associated α_i and *source* needed to construct the simplex S_+ should $\mathbf{v}_i$ produce the least function value, we use the simple algorithm shown in Table 3. To see why this algorithm works, we begin by noting that each of the core reflection, contraction, and expansion vertices are defined as follows:

$$\mathbf{r}_j = \mathbf{v}_0 + (-1)(\mathbf{v}_j - \mathbf{v}_0),$$

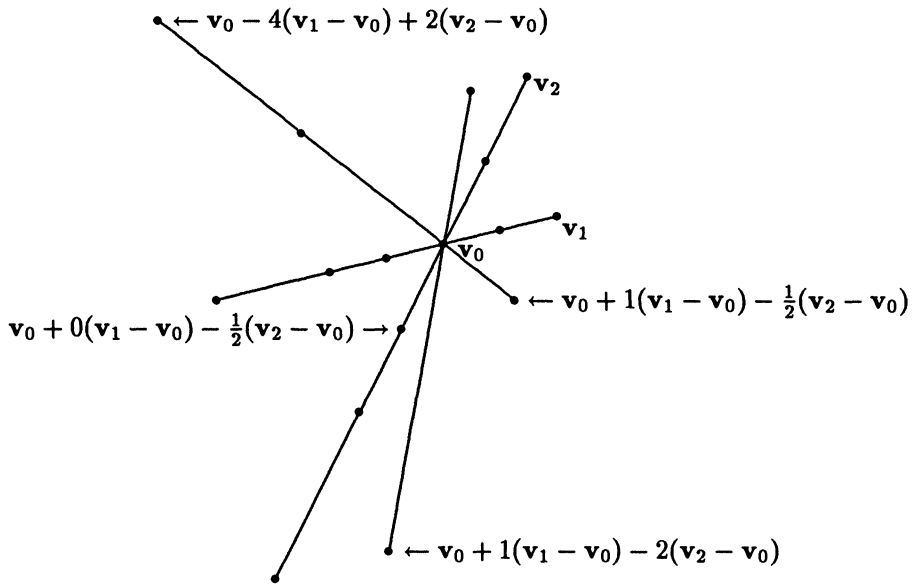


FIG. 9. *Defining the template.*

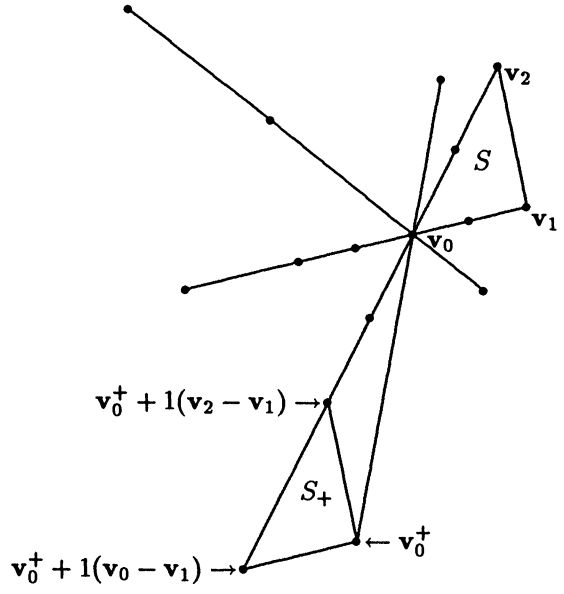


FIG. 10. *Constructing S_+ from $\mathbf{v}_0^+$ and S with $\alpha_+ = 1$ and source $= 1$.*

$$\begin{aligned} \mathbf{c}_j &= \mathbf{v}_0 + \left(\tfrac{1}{2}\right)(\mathbf{v}_j - \mathbf{v}_0), \quad \text{and} \\ \mathbf{e}_j &= \mathbf{v}_0 + (-2)(\mathbf{v}_j - \mathbf{v}_0), \end{aligned}$$

for $j = 1, \cdots, n$. Note that these definitions vary only in the choice of the scalars -1, $\frac{1}{2}$, and -2 associated with each step. We also know that we can construct S_+ from

TABLE 3
Initializing the template.

Given the reflection factor $\lambda = -1$, the contraction factor $\theta = \frac{1}{2}$,
and the expansion factor $\mu = -2$,
/* initialize the root of the tree by adding the current simplex */
$root \leftarrow 0$
$source_{root} \leftarrow 0$
$\alpha_{root} \leftarrow 1$
$coefficients_{root} \leftarrow \mathbf{0}$
$i \leftarrow 1$
repeat
/* generate all possible new best vertices given the current simplex */
    /* first consider all the new reflection vertices */
    for $j = 0, \cdots, n,\ j \neq source_{root}$
        $source_i \leftarrow j$
        $\alpha_i \leftarrow \lambda * \alpha_{root}$
        $coefficients_i \leftarrow coefficients_{root}$
        $coefficients_i(source_i) \leftarrow coefficients_i(source_i) + \alpha_i$
        $coefficients_i(source_{root}) \leftarrow coefficients_i(source_{root}) - \alpha_i$
        $i \leftarrow i + 1$
    end
    /* next consider all the new contraction vertices */
    for $j = 0, \cdots, n$
        $source_i \leftarrow j$
        $\alpha_i \leftarrow \theta * \alpha_{root}$
        $coefficients_i \leftarrow coefficients_{root}$
        $coefficients_i(source_i) \leftarrow coefficients_i(source_i) + \alpha_i$
        $coefficients_i(source_{root}) \leftarrow coefficients_i(source_{root}) - \alpha_i$
        $i \leftarrow i + 1$
    end
    /* finally consider all the new expansion vertices */
    for $j = 0, \cdots, n,\ j \neq source_{root}$
        $source_i \leftarrow j$
        $\alpha_i \leftarrow \mu * \alpha_{root}$
        $coefficients_i \leftarrow coefficients_{root}$
        $coefficients_i(source_i) \leftarrow coefficients_i(source_i) + \alpha_i$
        $coefficients_i(source_{root}) \leftarrow coefficients_i(source_{root}) - \alpha_i$
        $i \leftarrow i + 1$
    end
    $root \leftarrow root + 1$
until enough points have been generated

S, $\mathbf{v}_0^+$, α_+, and *source* as

$$\mathbf{v}_j^+ = \mathbf{v}_0^+ + \alpha_+ (\mathbf{v}_j - \mathbf{v}_{source}),$$

for $j = 0, \cdots, n$. Thus, given $\mathbf{v}_0^+$, α_+, and *source* we can construct the core step associated with $\mathbf{v}_0^+$. For instance, the reflection vertices would be

$$\begin{aligned} \mathbf{r}_j^+ &= \mathbf{v}_0^+ + (-1)\left(\mathbf{v}_j^+ - \mathbf{v}_0^+\right) \\ &= \mathbf{v}_0^+ + (-1)\left(\left(\mathbf{v}_0^+ + \alpha_+ (\mathbf{v}_j - \mathbf{v}_{source})\right) - \mathbf{v}_0^+\right) \\ &= \mathbf{v}_0^+ + (-1)(\alpha_+)(\mathbf{v}_j - \mathbf{v}_{source}) \\ &= \mathbf{v}_0^+ + (-1)(\alpha_+)\left((\mathbf{v}_j - \mathbf{v}_0) - (\mathbf{v}_{source} - \mathbf{v}_0)\right) \end{aligned}$$

for $j = 0, \cdots, n$, $j \neq$ *source*—which is exactly the representation we are using to construct the template.

We also note that there is additional work to be done once the list of coefficients has been generated, since there are duplicate coefficient vectors. We simply sort the list and then eliminate duplicates. (We also include the $n+1$ vertices in the original simplex when we check for duplicates so that they are not redefined.) Preference, in terms of the α and *source* associated with each coefficient vector, goes to the first definition. Note also that if we use $\theta = \frac{1}{2}$ and $\mu = -2$ (standard choices for algorithms of this sort), then we can scale the entire procedure by an appropriately large multiple of 2 and generate the template using integer arithmetic, which is faster, halves the required storage, and eliminates round-off error.

6.3. Stacking computation. The last claim we must substantiate is that it is easy to balance the cost of the computation versus the cost of the communication, or some other form of remote memory access. When we first began testing the basic multidirectional search algorithm on a shared memory multiprocessor, memory access completely swamped any gains to be seen from computing function values—at least those from the standard test set—in parallel. One way to overcome this imbalance would be to assume that the function evaluations are expensive enough to justify the use of a parallel machine. However, this imbalance is even more acute on the iPSC/860. If we had to assume that the function evaluations are expensive in order to justify using a parallel machine, then as processors become ever faster, the class of problems we would be able to consider solving on a parallel machine would become ever smaller.

The advantage of the modification we now introduce to the parallel direct search schemes is that while it introduces more computation on the individual processors, this additional computation need not have an appreciable effect on the execution time of the algorithm. We are simply trying to balance the cost incurred due to the global communication calls. Our modification is simple. Rather than assuming that the function evaluations are expensive, we simply construct more vertices on each processor and compute their associated function values before we synchronize the search. This simple modification is given in Table 4. As we shall see in the next section, this "extra" work is not wasted; in fact, it may actually lead to a significant *decrease* in the total execution time of the algorithm since the more ambitious search strategy that results may lead to significantly fewer iterations.

We now have all the ingredients we need to claim that, given the number of processors, the dimension of the problem, and some ratio of the cost of communication to the cost of computing the function value, we can tailor a parallel multidirectional

TABLE 4
The stacked parallel multidirectional search algorithm.

Given an initial simplex S_0 with vertices $\langle \mathbf{v}_0, \mathbf{v}_1, \cdots, \mathbf{v}_n \rangle$,
  initialize template
  while (stopping criterion is not satisfied) **do**
    for $i = 1, \cdots, p$ **do**
      for $j = 1, \cdots, k$ **do**
        $\mathbf{v}_j^i \leftarrow \mathbf{v}_0 + a_j^i(\mathbf{v}_1 - \mathbf{v}_0) + \cdots + z_j^i(\mathbf{v}_n - \mathbf{v}_0)$
        $fv_j^i \leftarrow f(\mathbf{v}_j^i)$
      end
      $fv_i \leftarrow \min_j \{fv_j^i\}$
    end
    $fv_* \leftarrow \min_i \{fv_i\}$ /* communication */
    update simplex
  end

search scheme to solve the particular problem in the given computing environment. Thus we effectively remove two issues that have plagued the parallelization of other optimization algorithms: the dimension of the problem, n, and the relative expense of function evaluations.

The remaining question is then: How well do these algorithms perform? We turn to the next section for some preliminary numerical results.

7. Numerical results. We wish to demonstrate two important features of the parallel direct search methods. First, these algorithms scale almost perfectly in the sense in which "scale" is usually applied to parallel computation: if we double the number of processors we use, we essentially halve the execution time of the algorithm. In other words, we have almost perfect linear speed-up in the performance of the algorithm.

However, our parallel direct search algorithms also scale in a way that is not so usual: not only can we increase the number of processors, we can also increase the number of points we compute on each processor before we synchronize the search by making a global communication call. This is the stacked parallel multidirectional search algorithm given in Table 4. When we fix the number of processors and increase the number of points we compute on each processor we are defining a new search strategy; we have a search pattern where the number of points in the search pattern is equal to the number of processors times the number of points computed on each processor before each global communication call. When we double the number of points in the search strategy, the decrease in execution time can be dramatic. This effect on the execution time, as we shall see, is highly nonlinear; with the right choice of strategies one may even have a better *sequential* algorithm since the total number of function evaluations required to converge to a solution will be less, even if one were computing more function evaluations at each iteration.

To demonstrate these two points, we will "borrow" the level curves of a classic test problem, Rosenbrock's function [17]:

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 .$$

We have borrowed the level curves because Rosenbrock's function is a function that

causes most optimization problems some difficulty when the search begins at the standard starting point $(-1.2, 1)$ in the sense that fifty iterations is fairly typical of better methods.

To make the computation more realistic, particularly given the speed of the processors on the iPSC/860, we have added 10,000 extra floating point operations to each function evaluation. This is why we say we have "borrowed" the level sets of Rosenbrock's function. It is worth noting that when we added 100,000 extra floating point operations to each function evaluation the execution times did increase but the observations we now report were qualitatively the same. We chose the lesser number of extra floating point operations to demonstrate that the relative expense of the function evaluation does have an effect on the efficiency of a parallel implementation.

We tested our parallel direct search method using most of the minimization problems found in Moré, Garbow, and Hillstrom [12]. We have singled out Rosenbrock's function because it gives fairly typical results; we observed similar behavior when we experimented with other problems in the Moré, Garbow, and Hillstrom test set. Later we will discuss our experience with some "real" problems.

We started each search at the classic starting point $(-1.2, 1)$. Since we require a simplex to start the search, we used a straightforward procedure found in [11] to generate a regular simplex with edges of length one. We stopped the search when the absolute value of the function (at the best vertex $\mathbf{v}_0$) fell below 10^{-7}, a decrease of eight orders of magnitude. There is nothing special about this choice; the algorithm ran equally well for choices between 10^{-1} and 10^{-10}. (Note that the stopping test we usually employ is a little more sophisticated [21]; we resorted to this naive choice for simplicity in discussing the efficiency of the algorithm in achieving some meaningful, well-defined goal.)

We then solved the problem varying two parameters: the number of processors and the number of points in the search pattern. The results can be seen in Figs. 11 and 12.

In Fig. 11, the linear speed-up in the execution time is apparent. There are plots for six different search strategies involving search patterns of 32, 64, 128, 256, 512, and 1024 points. We have plotted the $\log_2$ of the execution time in milliseconds as a function of the number of processors we have used. If we have linear speed-up, then as we double the number of processors, the execution time should be halved. The solid lines that bracket our plots in Fig. 11 demonstrate the slope we should see for perfect linear speed-up. And, in fact, we see essentially linear speed-up for all but two of the plots. The plots for the 32 point search pattern and the 64 point search pattern do show some degradation in the speed-up when we use all 32 processors, but this is easily explained. Even with over 10,000 floating point operations per function evaluation, we are not doing enough floating point operations to offset the overhead incurred due to the communication if we are only doing one or two function evaluations per processor before each global communication call.

In Fig. 12, the nonlinear behavior associated with the *algorithmic* changes is apparent. Here there are plots for six different choices in the number of nodes used to solve the problem. We have plotted the $\log_2$ of the execution time in milliseconds as a function of the number of points in the search pattern. For this particular example the best choice is 256 points. Note, however, that a 64 point search strategy is better than a 128 point search strategy and a 512 point search strategy is better than a 1024 point search strategy. But *any* of the more ambitious strategies takes less time to execute than the conservative 32 point strategy—*even on a single processor.*

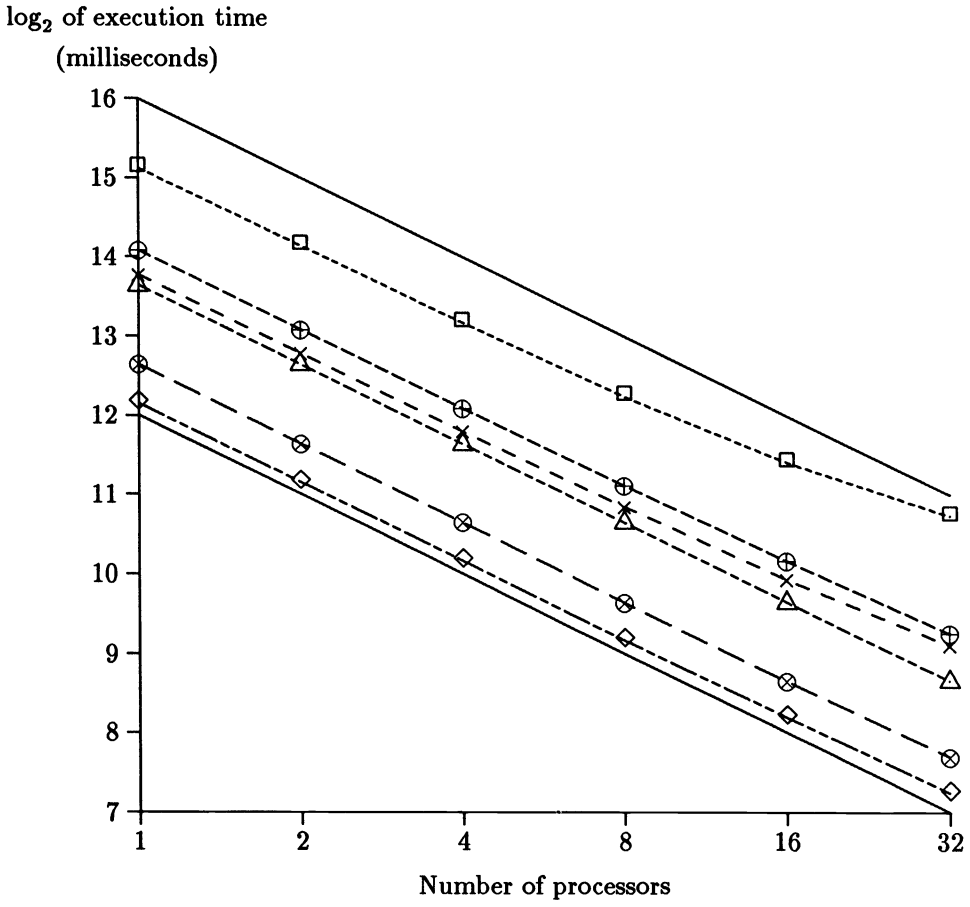


FIG. 11. *Linear speed-up obtained by doubling the number of processors.*

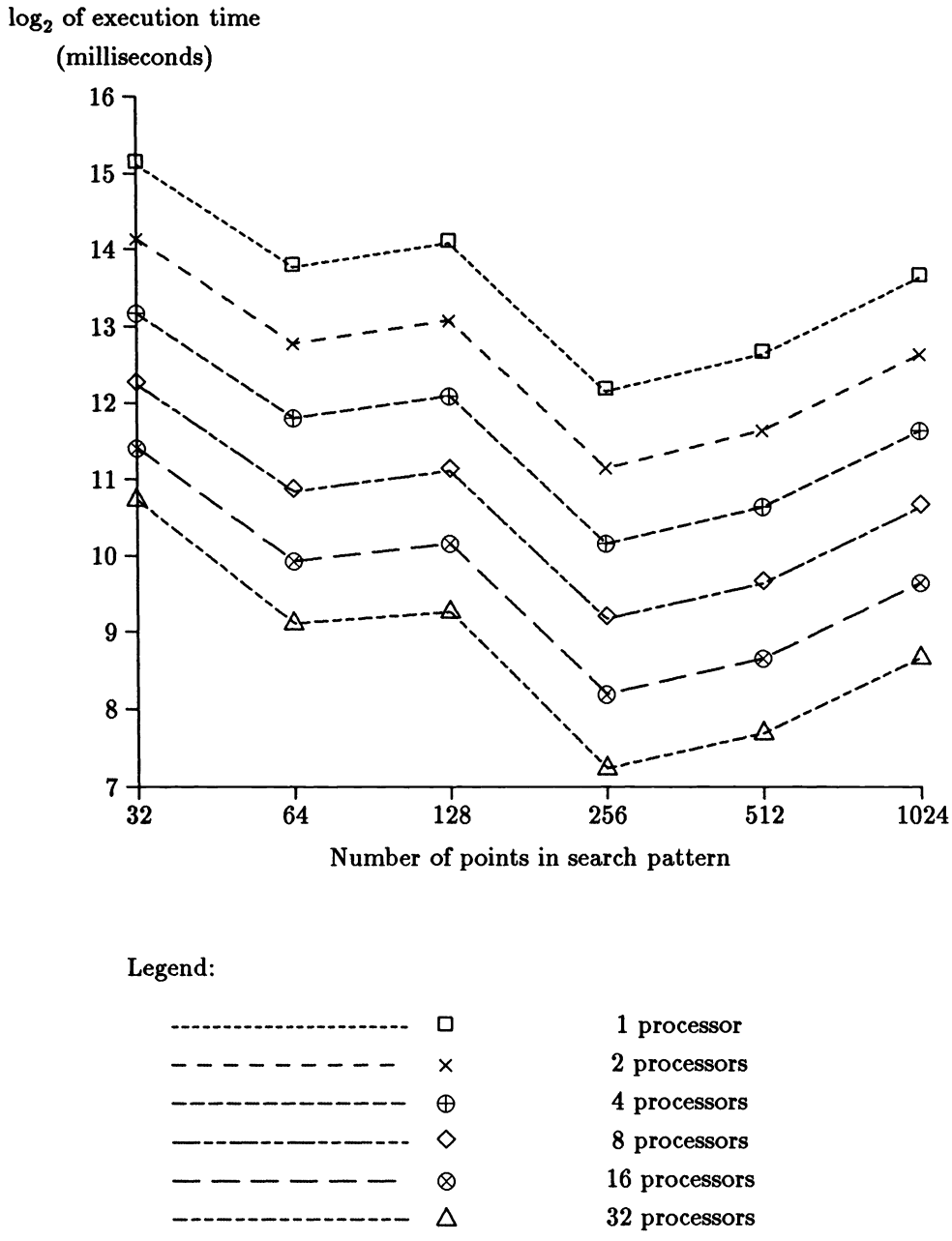


FIG. 12. *Nonlinear speed-up obtained by doubling the number of points in the search pattern.*

The "best" choice of search strategies will vary depending on the test problem, the starting point, the size and orientation of the initial simplex, etc. It is clear that there is a certain art involved in choosing the "optimal" number of points to be used in the search pattern. However, it is also true that in general the more points used in the search strategy, the better the information obtained at each iteration and thus the better the choice of new best vertices is likely to be. Again we stress that this improvement was seen across all the problems we tested.

Furthermore, even if we do not know a priori which search strategy is "best," we can still decrease the total execution time simply by adding more processors. Returning to Fig. 12, we see that a 256 point search strategy is optimal, regardless of the number of processors we use. However, *any* of the search strategies, if implemented on at least eight processors, will execute at least as quickly as the 256 point search strategy on a single processor. Thus, even if we do not know the "optimal" search strategy, we can still expect to see a decrease in the execution time simply by increasing the number of processors we use. If we wish to solve a problem repeatedly, it may be worthwhile to spend some time identifying an "optimal" search strategy. Otherwise, we might simply use as many processors as we can find (or afford).

Given the speed of the processors on the iPSC/860, we do not even require very many processors to undertake a more ambitious strategy. We solved a parameter identification problem in three unknowns that models catalytic cracking of gasoil to gasoline [8] in just over two seconds using a 32 point search strategy on eight processors. Each function evaluation required the numerical solution of a system of ordinary differential equations. Using four processors with the same 32 point search strategy took over four seconds; two processors required between eight and nine seconds. Again, we observed the linear behavior with respect to the number of processors.

Our tests of the parallel direct search methods lead us to two conclusions. First, these algorithms do scale almost perfectly in the usual sense: as long as we require a reasonable amount of computation on each processor, the communication requirements are so minimal that we see almost perfect linear speed-up in the performance of the algorithm, regardless of the problem we solve. If we double the number of processors we halve the execution time of the algorithm.

Another result is, at present, less well understood and less predictable. If we increase the number of points in the search strategy, which often involves a significant increase in the number of function values we compute (stack) on each node before we attempt to synchronize, we may actually see a marked decrease in the total execution time of the algorithm. These improvements have been dramatic for the more difficult problems.

It is important to understand that our parallel direct search methods do *not* place an upper bound on the number of processors that can be used. Given ever more processors we expect to be able to solve ever larger problems. We also note that much of the algorithmic improvement depends on the proper choice of a search pattern and there are many different ways to generate search patterns, even for a fixed number of points. As we gain more experience with these methods, it is possible that if we can define better search strategies, then we can further increase the range of problems that can be solved efficiently.

Finally, we note that the simplicity of the parallel direct search schemes suggests that they are very useful as experimental tools. All that is needed is a function evaluation subroutine. While one is calculating the derivatives, coding a subroutine, and then debugging the code to use a more sophisticated optimization method on a "real"

problem where the solution is not known, it is possible to be running experiments using the parallel direct search methods to determine reasonable starting points for the more sophisticated optimization method, as well as getting a feel for the general behavior of the function.[4]

The simplicity of the parallel direct search methods also means that they are less likely to fall prey to the pathologies, such as noise or the lack of continuity, that can plague more sophisticated optimization methods. As further evidence of this, we point to the success reported by Higham [9] using a sequential implementation, in Matlab, of the basic multidirectional search algorithm to investigate the stability and accuracy of algorithms in matrix computations. Higham observes that many of the questions of interest can be expressed in terms of some easily computable function f. The catch is that the function f is usually not smooth and derivatives, when they exist, are difficult to obtain. Thus, quasi-Newton or conjugate gradient methods are not applicable. Direct search methods, on the other hand, prove to be useful experimental tools.

We also have had experience experimenting with a data set provided by researchers at the University of Texas at Houston, M. D. Anderson Cancer Center, and the University of Texas Medical Branch at Galveston who are trying to derive mathematical models for the predictive value of early CA125 serum levels in epithelial ovarian carcinoma [1]. They have a model with five parameters that they originally fitted using NL2SOL [5]. They were concerned that NL2SOL required almost 900 function evaluations to return a solution. Experimenting with our parallel direct search methods, by successively restarting the optimization procedure, we were able to uncover that one of the parameters, which is required to be strictly greater than zero, was tending toward zero while another parameter, which is unbounded, was marching steadily towards $-\infty$—information that was not obvious from the solution returned by NL2SOL. They are now interested in further experiments to try to learn more about the behavior of their model.

The point here is not that the parallel multidirectional search algorithms produce optimal schemes for all problems on today's machines. Rather, when the problem to be solved is small, but difficult, and only a few significant digits in the solution are either required or expected, then the parallel direct search methods provide a simple and surprisingly effective approach. Furthermore, these methods may prove to be even more useful as experimental tools.

8. Future work. The next step is to try the parallel multidirectional search schemes on an inverse problem in multidimensional wave propagation. This problem can be formulated via coherency optimization as a low- (e.g., three-) dimensional minimization problem [19], [20]. Currently, an implementation of the objective function on a Stardent Titan takes, on average, several hours to return a function value. There is a tremendous amount of noise in the data so that the function values cannot be trusted to more than two digits. This means that finite-difference approximations to the gradient are really not feasible—and that an answer is expected to be accurate to only one or two digits. Thus, a quasi-Newton method seems out of the question and a direct search method seems to be in order.

[4] We had an answer for the parameter identification problem less than fifteen minutes after receiving the code for the objective function. We spent most of that time reading the accompanying instructions, compiling the code, and then entering the appropriate data. It took the iPSC/860 just over two seconds to return a solution *on our first try.* Writing a routine to evaluate the derivatives using finite-differences took thirty minutes and required some finesse. The solutions were equivalent.

Tackling this problem, however, means that we will need to rethink the original implementation of our parallel multidirectional search schemes. To begin with, our current implementation is best suited for the case when all the function evaluations require approximately the same time to complete. Thus, there is a natural synchronization that allows us to implement the algorithm without either a controlling process or any concerns for load balancing. This will not always be the case when dealing with more difficult problems. Hence, we will need an asynchronous, task-queue–based implementation with a single controlling process.

Another direction of research would be to extend the parallel multidirectional search algorithm to problems with constraints. We believe it is possible to extend the parallel algorithms, with only minor modifications, to problems with bounded variables. We are also interested in handling linear constraints. If we can handle bounded variables, it should be possible to transfer many of the ideas learned during the development of interior point methods to our simplex-based method for handling problems with linear constraints.

There are several other ideas we would also like to pursue. Although we have a simple, fast algorithm to generate templates for the parallel multidirectional search schemes, it is possible that there are other, perhaps better, initialization schemes we could implement.

One of the few pieces of information that the basic multidirectional search algorithm carries from iteration to iteration is the size of the step taken in the previous iteration—which determines the size of the step taken in the current iteration. If an expansion step was accepted, this would indicate that the simplex is still far from a solution. If the contraction step was accepted, then either the simplex is near a solution or it is trapped in a difficult region. If we allowed mixed strategies, i.e., different templates depending on the type of step accepted in the previous iteration, then it seems possible that we could further accelerate the search. This is an idea we plan to pursue further.

Currently we place no restrictions on the simplex used to start the procedure. The default option generates a regular simplex (i.e., a simplex with edges of equal length). Another standard option would be to start with a right-angle simplex: given a starting vertex $\mathbf{v}_0$, the remaining n vertices in the simplex are generated using the following simple formula:

$$\mathbf{v}_j \leftarrow \mathbf{v}_0 + \alpha_j \mathbf{e}_j ,$$

for $j = 1, \cdots, n$, where α_j is a nonzero scalar and $\mathbf{e}_j$ is the standard unit coordinate vector. If we were to restrict our attention to right-angle simplices, then we would only require two n vectors to store the entire simplex. Furthermore, producing the trial vertices for the search and updating the simplex would be reduced from $O(n^2)$ operations to $O(n)$. We plan to adopt this restriction when we implement the asynchronous version of our algorithm.

There is also the possibility that if the function values are not very expensive to calculate, and the processors are very fast, then the number of function evaluations we would need to stack on each processor, k, could also become quite large. However, as we noted during our discussion of the initialization, the template is generated using integer arithmetic. The information is then rescaled before being sent out to each processor. We could, instead, store the template in its integer form, which would halve the storage requirements, and simply perform the scaling required each time the information is used. A choice then has to be made as to which is the most efficient option.

Acknowledgments. We are grateful to the referees for their useful comments. We thank Robert Michael Lewis for his valuable suggestions on how best to present this material, particularly the results given in §7.

REFERENCES

[1] E. N. ATKINSON, M. G. DOHERTY, R. S. FREEMAN, AND H. A. FRITSCHE, *Mathematical models for the predictive value of early* CA125 *serum levels in epithelial ovarian carcinoma*, working paper.

[2] G. E. P. BOX, *Evolutionary operation: A method for increasing industrial productivity*, Appl. Statist., VI (1957), pp. 81–101.

[3] R. H. BYRD, R. B. SCHNABEL, AND G. A. SHULTZ, *Using parallel function evaluations to improve Hessian approximations for unconstrained optimization*, Tech. Report CS-CU-361-87, Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO, 1987.

[4] ———, *Parallel quasi-Newton methods for unconstrained optimization*, Math. Programming, 42 (1988), pp. 273–306.

[5] J. E. DENNIS, JR., D. M. GAY, AND R. E. WELSCH, *Algorithm* 573 *NL2SOL—an adaptive nonlinear least-squares algorithm* [E4], ACM Trans. Math. Software, 7 (1981), pp. 369–383.

[6] L. C. W. DIXON, *The place of parallel computation in numerical optimisation* I, *the local problem*, Tech. Report 118, Numerical Optimisation Centre, The Hatfield Polytechnic, Hatfield, Hertfordshire, U.K., 1981.

[7] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization*, Comput. J., 6 (1963), pp. 163–168.

[8] G. F. FROMENT AND K. B. BISCHOFF, *Chemical Reactor Analysis and Design*, John Wiley & Sons, New York, 1979.

[9] N. J. HIGHAM, *Optimization by direct search in matrix computations*, Tech. Report 197, Department of Mathematics, University of Manchester, Manchester, U.K., 1991.

[10] R. HOOKE AND T. A. JEEVES, *"Direct search" solution of numerical and statistical problems*, J. Assoc. Comput. Mach., 8 (1961), pp. 212–229.

[11] S. L. S. JACOBY, J. S. KOWALIK, AND J. T. PIZZO, *Iterative Methods for Nonlinear Optimization Problems*, Prentice–Hall, Englewood Cliffs, NJ, 1972.

[12] J. J. MORÉ, B. S. GARBOW, AND K. E. HILLSTROM, *Testing unconstrained optimization software*, ACM Trans. Math. Software, 7 (1981), pp. 17–41.

[13] S. G. NASH AND A. SOFER, *Block truncated-Newton methods for parallel optimization*, Math. Programming, 45 (1989), pp. 529–546.

[14] ———, BTN*: Software for parallel unconstrained optimization*, Tech. Report 64, Center for Computational Statistics, George Mason University, Fairfax, VA, 1990.

[15] ———, *A general-purpose parallel algorithm for unconstrained optimization*, Tech. Report 63, Center for Computational Statistics, George Mason University, Fairfax, VA, 1990; SIAM J. Optimization, (1991), this issue, pp. 530–547.

[16] J. A. NELDER AND R. MEAD, *A simplex method for function minimization*, Comput. J., 7 (1965), pp. 308–313.

[17] H. H. ROSENBROCK, *An automatic method for finding the greatest or least value of a function*, Comput. J., 3 (1960), pp. 175–184.

[18] SCI, *Science Citation Index Annual* 1989, Institute for Scientific Information, Inc., Philadelphia, PA.

[19] W. W. SYMES, *Velocity inversion: A case study in infinite-dimensional optimization*, Math. Programming, 48 (1990), pp. 71–102.

[20] W. W. SYMES AND J. J. CARAZZONE, *Velocity inversion by coherency optimization*, Tech. Report 89-8, Department of Mathematical Sciences, Rice University, Houston, TX, 1989; To appear in *Proceedings of the Workshop on Geophysics Inversion*, J. B. Bednar, ed., Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991.

[21] V. TORCZON, *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*, Ph.D. thesis, Rice University, Houston, TX, 1989; Available as Tech. Report 90-7, Department of Mathematical Sciences, Rice University, Houston, TX.

[22] ———, *On the convergence of the multidirectional search algorithm*, SIAM J. Optimization, 1 (1991), pp. 123–145.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 475-486, November 1991

003

ON THE IMPACT OF AUTOMATIC DIFFERENTIATION ON THE RELATIVE PERFORMANCE OF PARALLEL TRUNCATED NEWTON AND VARIABLE METRIC ALGORITHMS*

L. C. W. DIXON†

Abstract. The sparse doublet method for obtaining the gradient of a function or the Jacobian of a vector will be described and contrasted with reverse automatic differentiation. Its extension, the sparse triplet method for finding the Hessian of a function, will also be described and the effect of using these within classic optimisation algorithms discussed.

Results obtained using a parallel implementation of sparse triplet automatic differentiation of a partially separable function on the Sequent Balance will be presented.

In this paper it is shown that:

- automatic differentiation can no longer be neglected as a method for calculating derivatives;
- sparse triplets provide an effective method that can be implemented in parallel for calculating the Hessian matrix;
- this approach can be combined effectively with the truncated Newton method when solving large unconstrained optimisation problems on parallel processors.

Key words. automatic differentiation, parallel computation, optimisation

AMS(MOS) subject classifications. 49, 65

1. Introduction. In this paper we will be mainly concerned with the design of algorithms for solving the unconstrained optimisation problem

$$\text{Min } f(x), \qquad x \in R^n, \tag{1.1}$$

but the arguments used when discussing this problem apply equally well to many other areas, including the solution of the nonlinearly constrained optimisation problem, sets of nonlinear ordinary differential equations, and nonlinear partial differential equations.

The main theme of this paper is that the advent of automatic differentiation and parallel computation radically alters the relative efficiencies of traditional solution methods.

To introduce this theme we will, in § 2, recall the algorithmic position in optimisation in 1985 just before the impact of automatic differentiation. By that date there was, of course, already some indication of the likely impact of parallel computation, but this is easy to isolate and will not be mentioned in that section.

In this paper we have considered problems with large full Hessian matrices. The effect of sparsity has not been considered.

Section 3 will be devoted to an introduction to the various approaches to automatic differentiation that have been proposed and a discussion of how these by themselves affect the relative efficiencies of algorithms.

Section 4 will recall some early results using parallel computation and discuss some of the advantages and disadvantages they imply.

* Received by the editors August 17, 1990; accepted for publication (in revised form) March 29, 1991. This paper was presented at the Symposium on Parallel Optimization 2, held at University of Wisconsin, Madison, WI, July 1990. This research was partially supported by United States Army grant DAJA45-87-C-0038.

† Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, Hertfordshire AL10 9AB, United Kingdom.

2. The unconstrained optimisation problem. Most efficient sequential algorithms for solving the unconstrained optimisation problem (1.1) can be posed as two-stage iterative algorithms

$$x^{(k+1)} = x^{(k)} + \alpha p^{(k)}, \tag{2.1}$$

where $x^{(k)}$ is an iterative sequence of estimates of the solution x^*, and where at each iteration a search direction $p^{(k)}$ is first selected and then a stepsize is selected.

It is well known that we can guarantee that such two-stage iterative methods will enter a region around a stationary point of $f(x)$ that satisfies

$$\|\nabla f(x)\| \leqq \varepsilon_0 \tag{2.2}$$

in a finite number of iterations provided we satisfy Wolfe's conditions [19], [20] at a regular subsequence of iterations. We will assume that these conditions are satisfied in all the algorithms discussed.

In this paper we will contrast four methods of selecting $p^{(k)}$ to solve the general problem (1.1) and also recall modifications that are available when (1.1) has the special structure

$$f(x) = \sum_{j=1}^{J1} s_j^2(x), \tag{2.3}$$

where $J1$ is the number of terms in the definition of the function.

The four algorithms we will mention for the general problems are:

(1) the modified Newton method,
(2) the variable metric method,
(3) the conjugate gradient method, and
(4) the truncated Newton method.

In one sense all these algorithms have a common basis, namely, that at a local solution x^* to (1.1) the gradient $\nabla f(x^*)$ is zero and the Hessian $\nabla^2 f(x^*)$ is positive semidefinite.

So in Newton's method the direction $p^{(k)}$ would be obtained by solving

$$\nabla^2 f(x^{(k)}) p^{(k)} = -\nabla f(x^{(k)}), \tag{2.4}$$

whilst in a modified algorithm a diagonal matrix D would be introduced to ensure that $p^{(k)}$ satisfies Wolfe's Condition I, as in (2.5):

$$(\nabla^2 f(x) + D) p^{(k)} = -\nabla f(x^{(k)}). \tag{2.5}$$

Such a method therefore involves the calculation of $\nabla f(x)$ and $\nabla^2 f(x)$ at each iteration, followed by the solution of a set of n equations for $p^{(k)}$. These two tasks are the major computations in such a method and other algorithms can be viewed as attempts to reduce these tasks. But we will show that it is precisely these tasks that are affected by automatic differentiation and parallel computation.

The assumption behind the variable metric algorithm is that the calculation of $\nabla^2 f(x)$ is most expensive and that it is therefore advantageous to replace it by an approximation $B^{(k)}$ where $B^{(k)}$ is a modification to $B^{(k-1)}$ designed to ensure that the quasi-Newton condition is satisfied, i.e.,

$$B^{(k)}(x^{(k)} - x^{(k-1)}) = \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}).$$

The set of equations

$$B^{(k)} p^{(k)} = -\nabla f(x^{(k)}) \tag{2.6}$$

is then solved or, alternatively, to avoid the cost of solving a set of equations, the inverse approximation $H^{(k)}$ is used so that $p^{(k)}$ is given by

$$p^{(k)} = -H^{(k)}\nabla f(x^{(k)}). \tag{2.7}$$

If n is large, (2.7) has the advantage of not requiring the solution of a set of equations, which will either require $O(n^3)$ (the theoretical results decreasing this power from n^3 to $n^{2.9}$, etc., are acknowledged but will not be discussed—they do not effect the argument of this paper) operations or the use of sparse matrix solvers, but has the disadvantage that while B can be given, the sparsity pattern of $\nabla^2 f(x)$, H, will usually be full.

There are, of course, many additional and beneficial modifications to the variable metric approach, of which the use of an LL^T decomposition of B is but one example. Each would naturally modify the operational analysis presented but not the final conclusions.

This comment does not apply, however, to those specialised versions of the variable metric method, that use the variable metric principle to approximate small subsections of B (Griewank and Toint [12]).

For large problems the need for a matrix store can be avoided by reverting to the Fletcher–Reeves [10] conjugate gradient algorithm (or one of its many variants) where $p^{(k)}$ is simply given by

$$p^{(k)} = -\nabla f(x^{(k)}) + \beta p^{(k-1)} \tag{2.8}$$

and β is such that if f were a quadratic function and the stepsize were chosen so that

$$\nabla f(x^{(k)} + \alpha p^{(k)})^T p^{(k)} = 0, \tag{2.9}$$

then the method would terminate in a finite number of steps. The simplicity of this method of calculating $p^{(k)}$ is balanced by a significant increase in the number of iterations and the need for a more sophisticated line search. There are a number of variations on the basic conjugate gradient method; each variation requires a small number of vector operations. In the analysis in this paper, a figure of 7 is used as a typical figure.

One of the major innovations of the early 1980s was the introduction of the truncated Newton algorithm [3], [4], [18], in which it was noted that in the early iterations it was unnecessary and computationally wasteful to solve (2.4) or (2.5) accurately at each iteration, but that they could be treated as the necessary conditions for the minimum of a quadratic function Q and that this minimised the conjugate gradient process using the analytic value of α that then satisfied (2.9). This inner conjugate gradient iterative process is then terminated when

(1) Q is sufficiently reduced,

(2) $\|\nabla Q\|_2$ is sufficiently reduced,

(3) $\|x - x^{(k)}\|_2 \geqq d$ where d is the diameter of the current trust region, or

(4) when a negative curvature direction is identified, and then a downhill step to the boundary of the trust region is taken [8].

The inner iteration has the property that Q is reduced at each iteration and $\|x - x^{(k)}\|$ increased. Two versions of the algorithm are in use, one stores $\nabla^2 Q = \nabla^2 f(x)$ and uses it to calculate $\nabla^2 Q p^{(k-1)}$ and thus update ∇Q in (2.8); the other avoids storing or calculating $\nabla^2 Q$ by setting

$$\nabla^2 Q p^{(k-1)} = \{\nabla f(x + hp^{(k-1)}) - \nabla f(x)\}/h \tag{2.10}$$

for an appropriate small value of h.

Many computational studies, for instance, Dixon and Price [8], have indicated that this algorithm is more efficient than either the conjugate gradient or variable metric algorithms for most values of n.

The performance of this method can be greatly improved when storing $\nabla^2 Q$ if a preconditioning matrix is introduced while solving the system (2.4) or (2.5) by conjugate gradients. Our experience is that it is much harder to construct a suitable preconditioner if (2.10) is used.

It is our experience (see Dixon and Maany [6]) that when so preconditioned, the number of inner iterations (IIT) is rarely much greater than five even when $n \simeq 3{,}000$, though there will be problems for which this is very optimistic.

It is obvious that all these methods depend upon the availability of $\nabla f(x)$ and that some also need $\nabla^2 f(x)$.

Before the introduction of automatic differentiation, optimisers had to either obtain a formula for $\nabla f(x)$ and $\nabla^2 f(x)$ or to use numerical approximation techniques. Let us suppose our objective function $f(x)$ can be calculated at $x^{(k)}$ using M simple arithmetic operations (+, −, *, /, Λ, exp, log, sin, cos, etc.); and that there are n independent variables. Then as M and n increase, the task of obtaining the analytic formula for $\nabla f(x)$ becomes increasingly harder and indeed soon outstrips the patience of most humans and the capability of most symbolic differentiation codes (Griewank [11]).

Simple divided difference schemes that estimate

$$\nabla f_i(x) = \{f(x + h\hat{a}_i) - f(x)\}/h \quad \text{where } (\hat{a}_i)_k = \delta_{ik} \tag{2.11}$$

or

$$\nabla^2 f_{ij}(x) = \{f(x + h\hat{a}_i + h\hat{a}_j) - f(x + h\hat{a}_i) - f(x + h\hat{a}_j) + f(x)\}/h^2 \tag{2.12}$$

are both dependent on using an appropriate value of h and are expensive.

Given $f(x)$, then $\nabla f(x)$ requires nM operations and $\nabla^2 f(x)$ requires a further $\frac{1}{2}(n)(n+1)M$ operations.

For codes using this approach to obtain $\nabla f(x)$ and $\nabla^2 f(x)$, one set of relative efficiencies can readily be calculated.

The following calculations indicate the dominant operations at each iteration for problems with full Hessian matrices.

(1) **Newton's method.**

(2.13)

Form $\nabla^2 f(x)$, $\nabla f(x)$, $f(x)$;	$\frac{1}{2}(n+1)(n+2)$ M operations
Solve (2.4);	$\frac{1}{6}n^3$ operations

(2) **Variable metric method**

(2.14)

(2.1) Form $\nabla f(x)$;	$(n+1)$ M operations
Update B;	cn^2
and solve (2.6);	$\frac{1}{6}n^3$ operations

(2.15)

(2.2) Form $\nabla f(x)$;	$(n+1)$ M operations
Update H;	cn^2 operations
and multiply (2.7);	n^2 operations

(3) **Conjugate gradient algorithm**

(2.16)

Form $\nabla f(x)$;	$(n+1)$ M operations
Calculate p;	$7n$ operations
and fairly accurate line search (Dixon [5]);	$12M$ operations.

(4) **Truncated Newton algorithm**

(4.1) Using $\nabla^2 Q$;

(2.17)
form $\nabla^2 f(x)$, $\nabla f(x)$; $\frac{1}{2}(n+1)(n+2)$ M operations
together with the calculation of an appropriate preconditioner
do IIT inner iterations; IIT $(n^2+7n+\text{NPR})$ operations
where NPR is the number of nonzeros in the preconditioner.
(Note typically IIT = 5, though there will be problems for which this is very optimistic.)

(2.18)
(4.2) Using (2.10)
Form $\nabla(x)f$; $(n+1)$ M operations
do IIT inner iterations; IIT $(nM+7n)$ operations
(Note typically IIT > 5.)

In examining and using such calculations we must remember that the number of iterations required by each method on a particular function often differs greatly, but that on most problems the variable metric method requires more iterations than the Newton method (Dixon [5]), the conjugate gradient requires even more, whilst the truncated Newton method usually requires less inner iterations in total than the number of iterations used by the conjugate gradient method. When M is large compared to n the computational cost at an iteration of each method is dominated by the cost of calculating $\nabla f(x)$ and $\nabla^2 f(x)$. This has led to the concept of an effective number of function evaluations (EFEs), i.e., the number of iterations multiplied by the number of M operations needed to solve a problem, being used to compare the relative performance of algorithms.

3. Automatic differentiation. One of the earliest descriptions of automatic differentiation is that by Rall [17]. Essentially automatic differentiation introduces a new algebra. There are two versions, known as forward and reverse automatic differentiation. The simpler forward approach can be easily implemented in Ada.

We introduce a new data type, which we term a doublet U, that consists of the $n+1$ values of

$$u, \frac{\partial u}{\partial x_i}, \qquad i = 1, \cdots, n;$$

then if, during the calculation of $f(x)$, we add two variables $w = u + v$, then in evaluating $f(x)$ and $\nabla f(x)$, we add two doublets $W = U + V$ where

$$W = U + V = \left(u+v; \frac{\partial u}{\partial x_i} + \frac{\partial v}{\partial x_i}, \quad i = 1, \cdots, n \right).$$

As a doublet is a set of $n+1$ numbers, then using the facilities of new data types and the overriding of operations, each such operation can be easily performed.

$$W = U - V = \left(u - v, \frac{\partial u}{\partial x_i} - \frac{\partial v}{\partial x_i}, \quad i = 1, \cdots, n \right),$$

$$W = U * V = \left(u * v, u\frac{\partial v}{\partial x_i} + v\frac{\partial u}{\partial x_i}, \quad i = 1, \cdots, n \right),$$

$$W = 1/U = \left(1/u, -1/u^2 \frac{\partial u}{\partial x_i}, \quad i = 1, \cdots, n \right),$$

$$W = \log U = \left(\log u, 1/u \frac{\partial u}{\partial x_i}, \quad i = 1, \cdots, n \right).$$

This can readily be extended to triplets where

$$T=\left(u, \frac{\partial u}{\partial x_i}, \frac{\partial^2 u}{\partial x_i\, \partial x_j}, \quad i \geqq j\right).$$

In practice, using these "full" doublets and triplets requires more operations than the divided difference formula in (2.11), (2.12), and we can therefore always expect simple forward automatic differentiation schemes to take more time than the divided differences approach. Experience shows, however, that these doublets and triplets are usually sparse even for functions with full Hessian matrices and that the infill in doublet W is the union of the infill of doublets U with V in addition, subtraction, and multiplication, and is identical to U in the simple functions.

We therefore implement sparse doublets and triplets, (Dixon, Maany, and Mohseninia [7]). These are particularly effective on partially separable functions (Griewank and Toint [12]), for instance, given a function of form (2.3) where each $s_j(x)$, say, involved $M/J1$ operations and NEL variables. Then the number of operations to calculate $\nabla f(x)$ would be less than

$$\sum_{j=1}^{J1} (\mathrm{NEL}) * M/J1 = \mathrm{NEL} * M,$$

and for $\nabla^2 f(x)$, would be less than $\frac{1}{2}(\mathrm{NEL})(\mathrm{NEL}+1) * M$.

To indicate how effective this can be even on a function that is not partially separable, we note the table of ratios for the Helmholtz energy function (introduced by Griewank [11] as a test function for automatic differentiation) and quoted by Dixon, Maany, and Mohseninia [7], in which the ratios of (the times for calculating the gradient and Hessian)/(time of calculating the function value) are given in Table 1.

TABLE 1

The performance of sparse forward automatic differentiation on the Helmholtz energy function.

Dimension	∇f by divided difference	∇f by sparse doublet	$\nabla^2 f$ by sparse triplet
5	6	1.68	33
10	11	3.20	49
20	21	5.27	46
30	31	5.55	41
40	41	5.69	46
50	51	5.95	48
60	61	6.23	48
200	201	7.23	
500	501	8.15	

On the well-known predator/prey ordinary differential equation (O.D.E.) test function, Parkhurst [16] gives the results quoted in Table 2 for calculating the Jacobian $J(x)$ for dimensions up to $n = 5{,}000$ using the sparse doublet package.

EXAMPLE. Lotka-Volterra Predator/Prey Model (Byrne and Hindmarsh [1]).

$$r_1 = \frac{\text{Evaluation of } (f(x), J(x)) \text{ using sparse doublets}}{\text{Evaluation of } (f(x)) \text{ using ordinary arithmetic}},$$

$$r_2 = \frac{\text{Evaluation of } (f(x)) \text{ using sparse doublets}}{\text{Evaluation of } (f(x)) \text{ using ordinary arithmetic}}.$$

TABLE 2
Performance of sparse doublet automatic differentiation on the predator/prey problem.

Dimension	50	200	800	1,800	3,200	5,000
r_1	52.39	54.10	51.14	50.42	49.67	50.19
r_2	5.05	4.74	4.73	4.75	4.72	4.77

From these tables we can conclude that the (cost of evaluating $\nabla f(x)$)/(cost of evaluating $f(x)$) in the sparse doublet framework is about 10 (i.e., NEL$\simeq 10$ for these problems) and that there is a secondary cost, namely, that it costs more to evaluate $f(x)$ within the sparse doublet framework than outside and this ratio is itself about 5. This factor of 5 implies that the overhead time of accessing link lists and using the sparse data structure is much greater than the arithmetic time for calculating $f(x)$.

It is, of course, true that a clever and patient user could write a gradient evaluation code for his particular function that would do at least as well as the sparse doublet implementation and yet not incur this overhead.

Griewank [7] demonstrates that it is possible using a reverse sweep through the function graph to evaluate $\nabla f(x)$ in less than $5M$ arithmetic operations.

It should be emphasised that this gain in the number of operations is balanced by a much greater storage requirement since in the reverse method all M operations have to be retained for the derivative calculations, while in the forward mode each operation is only accessed once.

Ada implementations of this algorithm and of the equivalent reverse Hessian evaluation have been written by Christianson [2] using the computational graph. The theoretical bound for the Hessian is $25nM$, but each value of $\nabla^2 f(x) \cdot p$ can be obtained in $5M$ operations. Christianson's results [2], shown in Table 3, are significantly better than this. Again, we should emphasise that the overheads associated with the use of the data structure required by this method greatly exceed the cost of an ordinary function evaluation.

The contrasting values for sparse forward and backward calculation are:

	Reverse	Sparse Forward
$\nabla f(x)/f(x)$	5	NEL
$\nabla^2 f(x)/f(x)$	$25n$	$\frac{1}{2}$NEL(NEL+1)
$J(x)/f(x)$	$5n$	NEL$\leqq$max number of nonzeros in a row of $J(x)$
$\nabla^2 f(x)p/f(x)$	10 (see [2])	2.NEL

and it is therefore very easy for the forward calculation to be cheaper for partially separable sparse Hessian and Jacobian calculations.

TABLE 3
Performance of reverse automatic differentiation on the Helmholtz Energy Problem. (Time in seconds.)

Helmholtz Energy Function	$f(x)$ ordinary	$f(x)$ by graph	$\nabla f(x)$ by graph	$\nabla^2 f(x)p$ by graph
n 20	—	7	8	16
50	11	53	52	120
100	40	320	300	660

If we now substitute these values in our operations count for the algorithms, we obtain:

(1) **Newton's algorithm**

Form $\nabla^2 f(x)$, $\nabla f(x)$;	$25nM$ or $\frac{1}{2}(\text{NEL}) * (\text{NEL}+1)M$
Solve (2.4);	$\frac{1}{6}n^3$

(2) **Variable metric**

(2.1)	Form $\nabla f(x)$;	$5M$
	Update B;	cn^2
	Solve (2.6);	$\frac{1}{6}n^3$
(2.2)	Form $\nabla f(x)$;	$5M$
	Update H;	cn^2
	Multiply (2.7);	n^2

(3) **Conjugate gradient**

Form $\nabla f(x)$;	$5M$
Calculate p;	$7n$
Line search;	$12M$

(4) **Truncated Newton**

(4.1)	Form $\nabla^2 f(x)$, $\nabla f(x)$;	$25nM$ or $\frac{1}{2}(\text{NEL}) * (\text{NEL}+1)M$ plus calculation of preconditioner
	do IIT inner iterations;	$\text{IIT}(n^2+7n+\text{NPR})$
(4.2)	Form $\nabla f(x)$;	$5M$
	do IIT inner iterations;	$\text{IIT}(5M+7n)$

(5) **Truncated Gauss Newton for (2.3)**

(5.1)	Form $J(x)$;	$\text{NEL} * M$
	plus the cost of calculating a preconditioner	
	do IIT inner iterations;	$\text{IIT}(\text{NNZ}+cn+\text{NPR})$
(5.2)	Form $f(x)$;	$5 * M$
	do IIT inner iterations;	$\text{IIT}(5M+cn)$

The change in the dominant part of the operations count is quite remarkable.

In a family of functions where M is a linear function of n the following changes occur:

(1) For Newton's method, the solution of the set of equations now dominates the calculation of the Hessian.

(2) For the variable metric method for full approximate matrices this will still be true for both types of variable metric methods once $n > 25$. Indeed the handling of the full matrices makes it unattractive to use these methods, $n > 50$, and experience (see Dixon and Price [8]) shows that it is more expensive than truncated Newton for even small values of n.

(3) For the conjugate gradient method the cost of the function evaluations needed in the more accurate line searches now completely dominates the calculation of the gradient. It is indeed probable that as the calculation of $\nabla f^T p$ is now relatively cheap it would be better to use a safeguarded cubic line search rather than the parabolic method on which the estimate of $12M$ was based.

(4) For the truncated Newton method the reduction in the cost of evaluating $\nabla^2 f$ typically makes the use of a preconditioner for solving the inner iterations even more desirable, especially when NEL is small. The calculation of $\nabla^2 f$ is then preferable to the repeated calculation of $\nabla^2 fp$. As a small value of NEL also implies that the Hessian matrix is sparse, the n^2 term in the inner iteration also reduces to NNZ, the number of nonzeros in the Hessian.

In practice, for large values of n our extensive computational experience shows that automatic differentiation increases the superiority of the truncated Newton approach.

If M is large compared to n^2, then the variable metric approach is probably still competitive for small values of n, but the additional iterations normally required for higher values of n make it more costly than the truncated Newton method as n increases.

In the least squares setting the calculation of J by sparse doublets is now so cheap, and the reduction in IIT by using a preconditioner so significant that the use of 5.1 is definitely preferable (Parkhurst [16]).

4. Parallel computation. At the NOC, parallel versions of unconstrained optimisation codes have been tested on DAP [9], Sequent [14], and transputer networks [13]. Our experience has been that with partially separable functions, especially if $J1$ is a multiple of P (the number of processors) and each subfunction contains $M/J1$ operations, then if M is large enough, we can obtain almost perfect speedup in the function and gradient calculations. In this operation a particular processor is given the values of the NEL variables needed to calculate $s_j(x)$ and returns the value of $s_j(x)$ and its derivatives. The data communication is therefore small compared with the calculation.

Typical times for the sparse triplet evaluation of $f(x)$, $\nabla f(x)$, $\nabla^2 f(x)$ on the Sequent [14] are shown in Table 4 for the Olsen square cavity problem [15].

TABLE 4
Performance of parallel automatic differentiation on the Sequent Balance.

	Number of processors used									
Elements	1	2	3	4	5	6	7	8	9	10
8	37	20	12	10	11	12	13	8	—	—
32	156	85	62	44	38	35	26	22	24	26
128	670	352	227	175	146	124	107	92	87	81

When using the DAP, there is of course the restriction that it is an SIMD machine and therefore each subfunction must involve an identical computation. Such a problem was obtained by transforming a partial differential equation into an optimisation problem and solving it by conjugate gradient and truncation Newton codes. As these results were obtained before we began to use automatic differentiation, analytic formulae for $\nabla s_k(x)$ and $\nabla^2 s_k(x)$ were provided on the appropriate processors.

Both conjugate gradient and truncated Newton codes were successfully implemented and the truncated Newton methods were usually more efficient when sufficient store was available (Dixon and Ducksbury [9]). Typical results are shown in Table 5.

Here it must be noted that the direct mapping between the finite elements used to express the problem and the processor architecture enabled the algorithms to proceed without significant data transfer even in the calculation of p and that once more the speedup due to the use of a parallel processor is very impressive.

Regrettably, we have been unable to obtain such good results for the linear algebra within the truncated Newton method on those problems where this operation is essentially a sparse matrix/vector multiplication, or on either the Sequent or the transputer network and on these, the communication costs have implied that there was little point in calculating p in parallel. Indeed the results for sparse matrix vector multiplication of dimensions 512 and 1107 within the cavity-driven flow problem on the Sequent are shown in Table 6. Almost perfect speedup is, however, obtained on the transputer network when the Hessian matrix is full.

5. Conclusions. In this paper we have attempted to demonstrate that the advent of automatic differentiation should radically alter our concept of the relative efficiency of variable metric to truncated Newton and conjugate gradient codes.

Effectively, there seems little place for the simple variable metric approach in unconstrained optimisation, except for relatively small values of n now that the calculation of the Hessian of a partially separable function can be calculated so accurately and cheaply.

TABLE 5
Performance of parallel conjugate gradient algorithm on the DAP.

		Parallel (DAP)		Sequential (Dec 1091)	
P	$N \times N$	Iterations	CPU time seconds	Iterations	CPU time seconds
121	11	46	1.06	46	11.7
441	21	85	1.61	83	18.8
961	31	125	2.18	125	58.1
1,681	41	205	3.32	167	137.1
4,096	64	339	5.17		
4,096	65	513	30.14		
4,096	70	1,013	58.01		
4,096	80	1,345	76.64		
4,096	100	1,875	106.40		
4,096	127	2,417	136.76		

TABLE 6
Performance of sparse matrix/vector products on the Sequent Balance.

	Number of processors									
Time	1	2	3	4	5	6	7	8	9	10
$n = 512$	3.80	2.40	2.20	2.10	2.08	2.10	2.11	2.12	2.2	2.4
1,107	9.50	6.00	5.00	4.5	4.5	4.3	4.1	4.4	4.4	5.2

Similarly, the conjugate gradient method suffers due to the need for a semiaccurate line search which implies that it is usually not competitive with the truncated Newton method when M is large.

Our analysis and experience indicate that for general functions the version of the truncated Newton method that calculates $\nabla^2 f(x)$ at each inner iteration is preferable to that using (2.10) due to the possibility of including a preconditioner that substantially reduces the number of inner iterations.

We should acknowledge that one of the referees strongly disagrees with this conclusion, stating that in his experience it is easy and efficient to precondition (2.10). This has not been our experience.

These conclusions are even further reinforced on parallel computers where the calculation of the partially separable function values and derivatives can be performed with almost perfect speedup. Unfortunately, the speedup of the linear algebra that might be expected by forming sparse matrix/vector products in parallel by rows is lost due to communication costs on the two MIMD systems we have used, though almost perfect speedup is obtained if the matrices are full.

REFERENCES

[1] G. D. BYRNE AND A. C. HINDMARSH, *Stiff* O.D.E. *solvers: A review of coming and current attractions*, J. Comput. Phys., 70 (1987), pp. 1-62.

[2] B. CHRISTIANSON, *Automatic Hessians by reverse accumulation*, Tech. Report No. 228, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, U.K., 1990; also through private communication.

[3] R. DEMBO AND T. STEIHAUG, *Truncated Newton method for large scale optimisation*, Math. Programming, 26 (1983), pp. 190-212.

[4] R. DEMBO, S. C. EISENSTAT, AND T. STEIHAUG, *Inexact Newton method*, SIAM J. Numer. Anal., 19 (1982), pp. 400-408.

[5] L. C. W. DIXON, *Nonlinear optimisation: A survey of the state of the art*, in Software for Numerical Mathematics, D. Evans, ed., Academic Press, New York, 1973.

[6] L. C. W. DIXON AND Z. A. MAANY, *The performance of the truncated Newton conjugate gradient algorithm in Fortran and Ada*, Tech. Report, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, U.K., 1989.

[7] L. C. W. DIXON, Z. A. MAANY, AND M. MOHSENINIA, *Automatic differentiation of large sparse systems*, in Proc. Internat. Federation on Automatic Control, Symposium on Dynamic Modelling and Control of National Economies, Edinburgh, Scotland, 1989; J. Econom. Dynamics Control, to appear.

[8] L. C. W. DIXON AND R. C. PRICE, *Truncated Newton method for sparse unconstrained optimisation using automatic differentiation*, J. Optim. Theory Appl., 60 (1989), pp. 261-275.

[9] P. DUCKSBURY AND L. C. W. DIXON, *Experience running optimisation algorithms on parallel processing systems*, presented as a plenary session at the 11th Internat. Federation of Information Processing Societies Conference on System Modelling and Optimisation, Copenhagen, Denmark, 1983.

[10] R. FLETCHER AND C. M. REEVES, *Function minimisation by conjugate gradients*, Computer J., 7 (1964), pp. 149-154.

[11] A. GRIEWANK, *On automatic differentiation*, in Mathematical Programming 1988, Kluwer Academic Publishers, Japan, 1988.

[12] A. GRIEWANK AND P. TOINT, *On the unconstrained optimisation of partially separable functions*, in Nonlinear Optimisation 1981, M. J. D. Powell, ed., Academic Press, New York, 1982.

[13] M. JHA, *An implementation of a conjugate gradient algorithm on a transputer network*, Tech. Report No. 232, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, U.K., 1990.

[14] M. MOHSENINIA, *Concurrent optimisation on the Sequent Balance* 8,000, Tech. Report No. 226, Numerical Optimisation Centre, Hatfield Polytechnic, Hatfield, U.K., 1989.

[15] M. D. OLSEN AND S. TUANN, *A study of viscous finite element solution methods for the Navier-Stokes equations*, Department of Civil Engineering Report 14, University of British Columbia, Vancouver, Canada, 1976.

[16] S. C. PARKHURST, *The use of automatic differentiation in the numerical approximation of stiff* O.D.E.*s*, presented at the 11th Conference on Ordinary and Partial Differential Equations, Dundee University, Dundee, Scotland, 1990.

[17] L. B. RALL, *Automatic differentiation—Techniques and applications*, Lecture Notes in Computer Science 120, Springer-Verlag, Berlin, New York, 1981.

[18] PH. L. TOINT, *Towards an efficient sparsity exploiting Newton method for minimization*, in Sparse Matrices and their Uses, I. S. Duff, ed., Academic Press, New York, 1981.

[19] P. WOLFE, *Convergence conditions for ascent methods*, SIAM Rev., 11 (1969), pp. 226–235.

[20] ———, *Convergence conditions for ascent methods* II, SIAM Rev., 13 (1971), pp. 185–188.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 487–500, November 1991

004

PARALLEL CONSTRAINT DISTRIBUTION*

M. C. FERRIS† AND O. L. MANGASARIAN†

Abstract. Constraints of a mathematical program are distributed among parallel processors together with an appropriately constructed augmented Lagrangian for each processor, which contains Lagrangian information on the constraints handled by the other processors. Lagrange multiplier information is then exchanged between processors. Convergence is established under suitable conditions for strongly convex quadratic programs and for general convex programs.

Key words. parallel optimization, augmented Lagrangians, quadratic programs, convex programs

AMS(MOS) subject classification. 90C25

1. Introduction.

We are concerned with the problem

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_1(x) \leq 0, \cdots, g_k(x) \leq 0 \end{aligned} \tag{1.1}$$

where f, $g_1, \cdots, g_k$ are differentiable convex functions from the n-dimensional real space $\mathbb{R}^n$ to $\mathbb{R}$, $\mathbb{R}^{m_1}, \cdots, \mathbb{R}^{m_k}$, respectively, with f being strongly convex on $\mathbb{R}^n$. Our principal aim is to distribute the k constraint blocks among k parallel processors together with an appropriately modified objective function. We then solve each of these k subproblems independently, share Lagrange multiplier information among the processors, and repeat. Other recently proposed decomposition methods and applications thereof can be found in [22], [8], [5], [21]. The key to our approach lies in the precise form of the modified objective function to be optimized by each processor. Considerable experimentation with various Lagrangian terms [3] has highlighted the difference between theoretical convergence and computational efficiency. We believe that we now have effective modified objectives for each processor that can best be described as augmented Lagrangian functions [19], [20], [1]. The modified objectives are made up of the original objective function plus augmented Lagrangian terms involving the constraints handled by the other processors. Computational experience on the Sequent Symmetry S–81 shared memory multiprocessor with constraint distribution for quadratic programs derived from a least-norm solution of linear programs, has been encouraging. This is described in §4 of the paper. Section 2 is devoted to the quadratic programming case for which we obtain the strongest convergence results in Theorem 2.1. Under the assumption of a strongly convex quadratic objective and linear independence of each of the distributed constraint blocks, the parallel constraint distribution (PCD) algorithm converges from any starting point for a solvable problem. The key to the convergence proof is to show that in the dual space, the proposed parallel constraint distribution algorithm is equivalent to a subsequentially convergent iterative method with stepsize proposed in [11, Algorithm 2.1] for which full sequential convergence has just recently been established [9], [4], [17]. In §3 we establish a weaker convergence of the PCD algorithm (Theorem 3.2) for the general convex program (1.1) with a strongly convex objective function. The method of proof in this

* Received by the editors October 19, 1990; accepted for publication (in revised form) March 22, 1991. This material is based on research supported by Air Force Office of Scientific Research grant AFOSR–89–0410 and National Science Foundation grants DCR–8521228 and CCR–8723091.

† Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706.

section is entirely different from that of §2, and relies on the Lipschitz continuity of the solution variables of each subproblem in the fixed Lagrangian multipliers obtained from the other subproblems (Lemma 3.1). Unfortunately, to establish convergence, we need to assume that the distance between successive values of the multipliers approaches zero. We believe this assumption may be considerably relaxed and probably eliminated if one uses ideas of nonlinear Jacobi relaxation [18] for solving nonlinear complementarity problems.

A word about our notation is appropriate now. For a vector x in the n-dimensional real space $\mathbb{R}^n$, x_+ will denote the vector in $\mathbb{R}^n$ with components $(x_+)_i := \max\{x_i, 0\}$, $i = 1, \cdots, n$. The standard inner product of $\mathbb{R}^n$ will be denoted either by $\langle x, y\rangle$ or $x^T y$. The Euclidean, or 2-norm, $(x^T x)^{\frac{1}{2}}$, will be denoted by $\|\cdot\|$. For an $m \times n$ real matrix A, signified by $A \in \mathbb{R}^{m\times n}$, A^T will denote the transpose. The identity matrix of any order will be given by I. The nonnegative orthant in $\mathbb{R}^n$ will be denoted by $\mathbb{R}^n_+$. We will use the convention that $s = (s_1, \cdots, s_k)$, with each s_i representing either a component of the vector s or a block of components of the vector s. The meaning should be clear from the context.

2. Parallel constraint distribution for quadratic programs. For simplicity, we consider a quadratic program with three blocks of inequality constraints. Routine extension to k blocks can be achieved by appropriate extension and permutation of subscripts. Equality constraints can also be incorporated in a straightforward manner. Consider, then, the problem

$$\begin{aligned} &\text{minimize} && c^T x + \tfrac{1}{2} x^T Q x \\ &\text{subject to} && A_l x \le a_l, \qquad l = 1, 2, 3, \end{aligned} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $Q \in \mathbb{R}^{n\times n}$, $A_l \in \mathbb{R}^{m_l \times n}$, $a_l \in \mathbb{R}^{m_l}$, and Q is symmetric and positive definite. Furthermore, let

$$A := \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix} \quad \text{and} \quad a := \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

At iteration i of the algorithm we distribute the constraints of this problem among three parallel processors ($l = 1, 2, 3$) as follows:

$$\begin{aligned} \underset{x_l}{\text{minimize}} \quad & c^T x_l + \frac{1}{2} x_l^T Q x_l + \frac{1}{2\gamma} \left[\sum_{\substack{j=1 \\ j\neq l}}^{3} \left\| \left(\gamma(A_j x_l - a_j) + p^i_{jl}\right)_+ \right\|^2 \right] + x_l^T r^i_l \\ \text{subject to} \quad & A_l x_l \le a_l, \end{aligned} \tag{2.2}$$

where γ is a positive number and p^i_{jl} and r^i_l, $j, l = 1, 2, 3$ are defined below in (2.20) and (2.21). We note that the p^i_{jl} play the roles of multipliers and in fact converge to the optimal multipliers eventually, while r^i_l replaces estimates of the multipliers by their most recent values obtained from each of the other subproblems (see (2.20)). Note that the objectives of the subproblems (2.2) are quadratic augmented Lagrangians [19], [20], [1] perturbed by the linear terms $x_l^T r^i_l$. The motivation of this reformulation is that in each subproblem some constraints are treated explicitly as constraints while the remaining ones are treated as augmented Lagrangian terms in the objective function.

The updating of the multipliers is done by solving the subproblems explicitly rather than with the traditional, and often slow, gradient updating scheme in the dual space of the augmented Lagrangian approach [1]. Hence our method does not use a gradient or a proximal point multiplier updating scheme. The key to the convergence of our algorithm, for the quadratic case, is the choice of the parameters p_{jl}^i and r_l^i in such a way that the PCD algorithm is equivalent to a convergent iterative matrix-splitting method [11], [9], [14], [17] for a symmetric linear complementarity problem in the dual variables of the problem. This choice is by no means unique and we have experimented computationally with a number of choices for the p_{jl}^i and r_l^i, which we report on in §4. We shall establish convergence of only one of our choices in this section of the paper, which may not necessarily be the best computationally. Further experimentation is needed to determine the best splitting. We now proceed to show how the parameters p_{jl}^i and r_l^i are chosen and to justify these choices from the point of view of convergent matrix splitting. A simpler splitting approach for constraint distribution for quadratic programs is given in [6].

First, note that it is easy to verify algebraically the following equivalence for any two vectors b and d in $\mathbb{R}^{m_l}$:

$$b = d_+ \iff b - d \geq 0, \qquad b^T(b-d) = 0, \qquad b \geq 0. \tag{2.3}$$

Also, the linear complementarity problem (LCP) in the variable z

$$Mz + q \geq 0, \qquad \langle z, Mz + q \rangle = 0, \qquad z \geq 0$$

can be written as

$$z = (z - Mz - q)_+$$

by using (2.3).

Now, let $(\bar{x}_l^{i+1}, \bar{s}_l^{i+1}) \in \mathbb{R}^{n+m_l}$, $l = 1,2,3$, $i = 1, \cdots$ satisfy the Karush–Kuhn–Tucker conditions [10] for subproblems (2.2). We shall signify this by

$$(\bar{x}_l^{i+1}, \bar{s}_l^{i+1}) \in \arg \mathrm{KKT}\,(2.2)\,.$$

Using (2.3), we see that $(\bar{x}_l^{i+1}, \bar{s}_l^{i+1})$ satisfy the following Karush–Kuhn–Tucker conditions:

$$\begin{aligned} &c + Q\bar{x}_l^{i+1} + \sum_{\substack{j=1 \\ j \neq l}}^{3} A_j^T \left(\gamma(A_j\bar{x}_l^{i+1} - a_j) + p_{jl}^i\right)_+ + r_l^i + A_l^T \bar{s}_l^{i+1} = 0 \\ &\bar{s}_l^{i+1} = \left(\bar{s}_l^{i+1} + \gamma(A_l\bar{x}_l^{i+1} - a_l)\right)_+ \end{aligned} \tag{2.4}$$

for $l = 1,2,3$ or equivalently

$$\begin{aligned} \bar{x}_l^{i+1} &= -Q^{-1}\left(c + \sum_{\substack{j=1 \\ j \neq l}}^{3} A_j^T \bar{t}_{jl}^{i+1} + r_l^i + A_l^T \bar{s}_l^{i+1}\right) && l = 1,2,3, \\ \bar{s}_l^{i+1} &= \left(\bar{s}_l^{i+1} + \gamma(A_l\bar{x}_l^{i+1} - a_l)\right)_+ && j = 1,2,3, \\ \bar{t}_{jl}^{i+1} &= \left(\gamma(A_j\bar{x}_l^{i+1} - a_j) + p_{jl}^i\right)_+ && j \neq l. \end{aligned} \tag{2.5}$$

Eliminating $\bar{x}_l^{i+1}$ by using the first equation of (2.5) leads to

$$
\begin{aligned}
\bar{s}_l^{i+1} &= \left(\bar{s}_l^{i+1} - \gamma\left(A_l Q^{-1}(A_l^T \bar{s}_l^{i+1} + \textstyle\sum_{\substack{k=1\\k\neq l}}^{3} A_k^T \bar{t}_{kl}^{i+1} + r_l^i + c) + a_l\right)\right)_+ \\
\bar{t}_{jl}^{i+1} &= \left(-\gamma\left(A_j Q^{-1}(A_l^T \bar{s}_l^{i+1} + \textstyle\sum_{\substack{k=1\\k\neq l}}^{3} A_k^T \bar{t}_{kl}^{i+1} + r_l^i + c) + a_j\right) + p_{jl}^i\right)_+
\end{aligned}
\tag{2.6}
$$

for $l = 1,2,3$, $j = 1,2,3$, $j \neq l$. Note that by (2.3), this is an LCP in the variable $\bar{z}^{i+1}$ defined by

$$
\bar{z}^{i+1} := (\bar{s}_1^{i+1}, \bar{s}_2^{i+1}, \bar{s}_3^{i+1}, \bar{t}_{12}^{i+1}, \bar{t}_{23}^{i+1}, \bar{t}_{31}^{i+1}, \bar{t}_{13}^{i+1}, \bar{t}_{21}^{i+1}, \bar{t}_{32}^{i+1}). \tag{2.7}
$$

In order to express the above LCP succinctly, we introduce the following notation. Define the permutations

$$
\sigma_1 = (1,2,3), \qquad \sigma_2 = (2,3,1), \qquad \sigma_3 = (3,1,2),
$$

with $\sigma_j(k)$ denoting the kth component of σ_j, $j,k = 1,2,3$. We use the following conventions to group p_{jl}^i, t_{jl}^i, and r_l^i:

$$
r^{i_j} := \begin{bmatrix} r_{\sigma_j(k)}^i \end{bmatrix}
$$

and

$$
p^{i_j} := \begin{bmatrix} p_{k,\sigma_j(k)}^i \end{bmatrix}, \qquad t^{i_j} := \begin{bmatrix} t_{k,\sigma_j(k)}^i \end{bmatrix}, \qquad j = 2,3, \qquad k = 1,2,3.
$$

(For example, $r^{i_2} = (r_2^i, r_3^i, r_1^i)$ and $t^{i_2} = (t_{12}^i, t_{23}^i, t_{31}^i)$.) Using this notation, (2.6) corresponds to the following symmetric LCP in the variable $\bar{z}^{i+1}$

$$
B\bar{z}^{i+1} + h^i + q \geq 0, \qquad \langle \bar{z}^{i+1}, B\bar{z}^{i+1} + h^i + q \rangle = 0, \qquad \bar{z}^{i+1} \geq 0 \tag{2.8}
$$

where

$$
B := \begin{bmatrix} R_1 & R_2^T & R_3^T \\ R_2 & I + R_1 & R_2^T \\ R_3 & R_2 & I + R_1 \end{bmatrix} \tag{2.9}
$$

with

$$
\begin{aligned}
R_1 &:= \gamma \begin{bmatrix} A_1 Q^{-1} A_1^T & 0 & 0 \\ 0 & A_2 Q^{-1} A_2^T & 0 \\ 0 & 0 & A_3 Q^{-1} A_3^T \end{bmatrix}, \\
R_2 &:= \gamma \begin{bmatrix} 0 & A_1 Q^{-1} A_2^T & 0 \\ 0 & 0 & A_2 Q^{-1} A_3^T \\ A_3 Q^{-1} A_1^T & 0 & 0 \end{bmatrix}, \\
R_3 &:= \gamma \begin{bmatrix} 0 & 0 & A_1 Q^{-1} A_3^T \\ A_2 Q^{-1} A_1^T & 0 & 0 \\ 0 & A_3 Q^{-1} A_2^T & 0 \end{bmatrix},
\end{aligned}
\tag{2.10}
$$

and

$$
h^i := \begin{bmatrix} \gamma A Q^{-1} r^{i_1} \\ \gamma A Q^{-1} r^{i_2} - p^{i_2} \\ \gamma A Q^{-1} r^{i_3} - p^{i_3} \end{bmatrix}, \qquad q := \gamma \begin{bmatrix} A Q^{-1} c + a \\ A Q^{-1} c + a \\ A Q^{-1} c + a \end{bmatrix}. \tag{2.11}
$$

(For the general case, the analog of these equations can be constructed easily by using the appropriate permutations of $1, \cdots, l$ and noting that the nonzero entries of the R_i correspond precisely to the ith permutation.) Note that this algorithm can be implemented in parallel in the x space as outlined at the start of this section for any choice of p_{jl}^i and r_l^i. In the remainder of this section, we show how to choose p_{jl}^i and r_l^i in order to guarantee the convergence of the algorithm. Our convergence analysis will be based on results for matrix-splitting methods for complementarity problems, and we give a very brief review of the pertinent results in the following paragraph.

A matrix-splitting method for solving the LCP

$$Mz + q \geq 0, \qquad \langle z, Mz + q \rangle = 0, \qquad z \geq 0 \tag{2.12}$$

uses a "regular splitting" $M = B + C$, with M, B, and C satisfying certain properties such as:

$$\begin{aligned} &B + C \text{ symmetric,} \\ &(1 - \lambda/2)B - (\lambda/2)C \text{ positive definite for some } \lambda \in (0, 1] \end{aligned} \tag{2.13}$$

or

$$\begin{aligned} &B + C \text{ symmetric positive semidefinite,} \\ &(1 - \lambda/2)B - (\lambda/2)C \text{ positive definite for some } \lambda \in (0, 1]. \end{aligned} \tag{2.14}$$

Under (2.13), a solution of (2.12) is obtained [11] from each accumulation point of the sequence $\{z^i\}$ generated by iteratively solving the following LCP for $\bar{z}^{i+1}$:

$$B\bar{z}^{i+1} + Cz^i + q \geq 0, \qquad \langle \bar{z}^{i+1}, B\bar{z}^{i+1} + Cz^i + q \rangle = 0, \qquad \bar{z}^{i+1} \geq 0 \tag{2.15}$$

and then determining z^{i+1} by using a stepsize λ, that is,

$$z^{i+1} = (1 - \lambda)z^i + \lambda\bar{z}^{i+1}, \qquad \lambda \in (0, 1]. \tag{2.16}$$

Under assumption (2.14) the *whole* sequence $\{z^i\}$ generated by (2.15) and (2.16) converges to a solution of (2.12) provided the latter is solvable [9], [4], [17].

To apply these results to our algorithm, we have to choose p_{jl}^i and r_l^i as particular functions of

$$z^i := (s_1^i, s_2^i, s_3^i, t_{12}^i, t_{23}^i, t_{31}^i, t_{13}^i, t_{21}^i, t_{32}^i) \tag{2.17}$$

so that

$$h^i = Cz^i, \quad \text{for some matrix } C \tag{2.18}$$

and

$$B + C \text{ constitutes a "regular splitting" of some symmetric } M.$$

The matrix C is determined by the choice of p_{jl}^i and r_l^i in (2.2) or equivalently in (2.11), and this is precisely where the power (and at the same time the difficulty) of the proposed method lies.

The simplest choice for p_{jl}^i and r_l^i that we propose for the nonlinear (not necessarily quadratic) case of §3 and for which we establish convergence under somewhat more stringent assumptions is the following:

$$p_{jl}^i = s_j^i, \qquad r_l^i = 0, \qquad l = 1, 2, 3, \qquad j = 1, 2, 3, \qquad j \neq l. \tag{2.19}$$

Unfortunately, this simple choice in the quadratic case leads to a *nonsymmetric* C and hence a nonsymmetric M in (2.12). The convergence conditions for splitting nonsymmetric LCPs are quite stringent [2, Chap. 5] and not useful for our proposed applications here.

We have therefore settled on choices for the parameters p_{jl}^i and r_l^i, which are different from those in (2.19), and which generate a symmetric positive semidefinite M. By choosing λ sufficiently small, it is easily seen that (2.14) is satisfied, because by (2.9), the matrix B is positive definite if we assume that each A_l, $l = 1, 2, 3$, has linearly independent rows. This can be shown by substituting for R_1, R_2, and R_3 from (2.10) into the definition of B. There are a number of choices of the p_{jl}^i and r_l^i that generate a symmetric positive semidefinite M and hence a convergent scheme. Our preliminary computational experience does not provide a clear-cut indication which is the best choice for p_{jl}^i and r_l^i among the convergent schemes. We believe this requires further theoretical and computational study. However, for concreteness, we wish to present at least one specific choice of C that results from the following choices of p_{jl}^i and r_l^i:

$$\begin{aligned} r_1^i &= A_2^T(s_2^i - t_{21}^i) + A_3^T(s_3^i - t_{31}^i), \\ r_2^i &= A_1^T(s_1^i - t_{12}^i) + A_3^T(s_3^i - t_{32}^i), \\ r_3^i &= A_1^T(s_1^i - t_{13}^i) + A_2^T(s_2^i - t_{23}^i); \end{aligned} \tag{2.20}$$

$$\begin{aligned} p^{i_2} &= t^{i_2} + \gamma A Q^{-1} A^T (s^i - t^{i_2}), \\ p^{i_3} &= t^{i_3} + \gamma A Q^{-1} A^T (s^i - t^{i_3}). \end{aligned} \tag{2.21}$$

We note that the r_l^i substitute the latest Lagrange multiplier value s_l^i obtained from each subproblem solution for the t_{jl}^i, both of which eventually converge to an optimal Lagrange multiplier value. The p_{jl}^i terms are essentially multiplier value estimates given by t_{jl}^i plus additional terms that converge to zero. The additional terms are added in order to produce a symmetric C and hence a symmetric M. The above choices of p_{jl}^i and r_l^i lead to the following matrix C, defined through the relations (2.17), (2.18), and (2.11):

$$C = \begin{bmatrix} R_2 + R_3 & -R_2^T & -R_3^T \\ -R_2 & -I + R_2 + R_3 & -R_2^T \\ -R_3 & -R_2 & -I + R_2 + R_3 \end{bmatrix} \tag{2.22}$$

with R_2 and R_3 defined in (2.10). Addition of the matrices B and C gives the symmetric block-diagonal matrix

$$M = \begin{bmatrix} H & 0 & 0 \\ 0 & H & 0 \\ 0 & 0 & H \end{bmatrix} \tag{2.23}$$

where

$$H := R_1 + R_2 + R_3 = AQ^{-1}A^T. \tag{2.24}$$

Note that if our original quadratic program (2.1) is feasible, then it it solvable. Hence its Wolfe dual is solvable, which is equivalent to the solvability of the LCP (2.12) with M as defined in (2.23) and q as in (2.11). In fact, the LCP (2.12) constitutes a replication of the Wolfe dual three times.

We are now ready to define the PCD algorithm for the quadratic program (2.1).

2.1. PCD algorithm for quadratic programming.
Initialization: Start with any s_l^0, t_{jl}^0, $l = 1,2,3$, $j = 1,2,3$, $j \neq l$.
Parallel iteration: In parallel, $(l = 1,2,3)$, perform the following steps.
Having s_l^i, t_{jl}^i, $j = 1,2,3$, $j \neq l$ compute:

1. r_l^i, p_{jl}^i, $j = 1,2,3$, $j \neq l$ from (2.20) and (2.21)
2. $(\bar{x}_l^{i+1}, \bar{s}_l^{i+1}) \in \arg \mathrm{KKT}\,(2.2)$
3. $\bar{t}_{jl}^{i+1} = \left(\gamma(A_j \bar{x}_l^{i+1} - a_j) + p_{jl}^i\right)_+$, $j = 1,2,3$, $j \neq l$
4. $(s_l^{i+1}, t_{jl}^{i+1}) = (1-\lambda)(s_l^i, t_{jl}^i) + \lambda(\bar{s}_l^{i+1}, \bar{t}_{jl}^{i+1})$, $j = 1,2,3$, $j \neq l$ with $\lambda \in (0,1]$ satisfying (2.30) below.

2.2. Remark. We note that the subproblems (2.2) of the PCD algorithm 2.1 divide the constraints of the original quadratic program (2.1) between them in the form of explicit constraints as well as augmented Lagrangian terms involving the remaining constraints. The principal objective that has been achieved is that the *explicit* constraints of each of the subproblems are a subset of the constraints of the original problem.

2.3. Remark: Symmetric monotone LCP as dual of convex quadratic program with nonsmooth KKT conditions. It is interesting to note that the PCD algorithm is a matrix-splitting iterative method for a symmetric monotone LCP that can be associated with a dual formulation of a convex program with nonsmooth Karush–Kuhn–Tucker conditions. Thus consider such a program:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & c^T x + \frac{1}{2} x^T Q x + \frac{1}{2} \left\| (Hx - h)_+ \right\|^2 \\ \text{subject to} & Bx \leq b, \end{array} \tag{2.25}$$

where Q is symmetric positive definite. The necessary and sufficient Karush–Kuhn–Tucker conditions for this problem are

$$\begin{aligned} & c + Qx + H^T (Hx - h)_+ + B^T s = 0, \\ & s = (s + Bx - b)_+ . \end{aligned} \tag{2.26}$$

Defining a new variable t as

$$t = (Hx - h)_+ \tag{2.27}$$

and solving the first Karush–Kuhn–Tucker condition for x gives

$$x = -Q^{-1}(H^T t + B^t s + c). \tag{2.28}$$

Substituting for x in the second equation of (2.26) and in (2.27) gives the following symmetric monotone linear complementarity problem in the variables (s,t)

$$\begin{aligned} & \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} BQ^{-1}B^T & BQ^{-1}H^T \\ HQ^{-1}B^T & I + HQ^{-1}H^T \end{bmatrix} \begin{bmatrix} s \\ t \end{bmatrix} + \begin{bmatrix} BQ^{-1}c + b \\ HQ^{-1}c + h \end{bmatrix} \geq 0, \\ & (s^T, t^T) \begin{bmatrix} v \\ w \end{bmatrix} = 0, \qquad \begin{bmatrix} s \\ t \end{bmatrix} \geq 0. \end{aligned} \tag{2.29}$$

We then have the following duality relation between the convex program (2.25) and the symmetric monotone LCP (2.29). For each solution (s,t) of (2.29), x defined by (2.28) is the unique solution of (2.25). Conversely, for each Karush–Kuhn–Tucker point (x,s) of (2.25), the point (s,t), with t defined by (2.27), solves (2.29). Note that the symmetric LCP (2.29) is equivalent to the following quadratic program in (s,t):

$$\underset{(s,t)\geq 0}{\text{minimize}} \frac{1}{2}(s^T,t^T)\begin{bmatrix} BQ^{-1}B^T & BQ^{-1}H^T \\ HQ^{-1}B^T & I+HQ^{-1}H^T \end{bmatrix}\begin{bmatrix} s \\ t \end{bmatrix} + (s^T,t^T)\begin{bmatrix} BQ^{-1}c+b \\ HQ^{-1}c+h \end{bmatrix}.$$

We are now ready to establish convergence of the PCD algorithm 2.1.

THEOREM 2.1 (PCD Convergence for Quadratic Programs). *Let* (2.1) *be feasible and let* Q *be symmetric positive definite and let each of* A_l, $l=1,2,3$, *have linearly independent rows. Then the sequence* $\{s_l^i, t_{jl}^i\}$, $l=1,2,3$, $j=1,2,3$, $j\neq l$, $i=0,1,\cdots$, *generated by the* PCD *algorithm converges to* $(\bar{s}_l,\bar{t}_{jl})$, $l=1,2,3$, $j=1,2,3$, $j\neq l$ *and each of the sequences* $\{x_l^i\}$, $l=1,2,3$, *converges to the unique solution* $\bar{x}$ *of* (2.1). *Furthermore,* $(\bar{x},\bar{s})$, $(\bar{x},\bar{t}_{12},\bar{t}_{23},\bar{t}_{31})$, *and* $(\bar{x},\bar{t}_{13},\bar{t}_{21},\bar{t}_{32})$ *are all Karush–Kuhn–Tucker points for* (2.1), *and* $\bar{p}_{jl}=\bar{t}_{jl}$, $l=1,2,3$, $j=1,2,3$, $j\neq l$.

Proof. Let $\bar{x}_l^{i+1}$, $l=1,2,3$, be the unique solution of the subproblems (2.2). Hence $\bar{x}_l^{i+1}$ and some $\bar{s}_l^{i+1}\in\mathbb{R}^{m_l}$ satisfy the Karush–Kuhn–Tucker conditions (2.4), or equivalently, $(\bar{x}_l^{i+1},\bar{s}_l^{i+1})$ and some $\bar{t}_{jl}^{i+1}$, $l=1,2,3$, $j=1,2,3$, $j\neq l$ satisfy (2.5). This in turn is equivalent to $\bar{z}^{i+1}$, as defined by (2.7), satisfying the LCP (2.15). By the choice of r_l^i, p_{jl}^i, $l=1,2,3$, $j=1,2,3$, $j\neq l$ of (2.20) and (2.21), it follows that the matrix $M=B+C$, given by (2.23) and (2.24), is symmetric and positive semidefinite. Furthermore, B is positive definite by virtue of the linear independence of A_l, $l=1,2,3$. Thus if λ is chosen sufficiently small, and specifically such that

$$0<\lambda\leq 1 \quad\text{and}\quad \lambda<2(\text{min eigenvalue}(B)\ /\ \text{max eigenvalue}(M)), \tag{2.30}$$

it follows that (2.13) (which is condition (6) of [11]) and (2.14) (which is condition (4.1) of [9]) are satisfied. Hence, since the LCP (2.12) is solvable, the sequence $\{z^i\}$ converges [9, Thm. 2 and Ex. 3] to a solution of the LCP (2.12), and by $z^{i+1}=(1-\lambda)z^i+\lambda\bar{z}^{i+1}$, so does the sequence $\{\bar{z}^i\}$. It follows by (2.4), (2.5), (2.20), and (2.21) that in the limit we have

$$\begin{aligned} &c+Q\bar{x}_l+\textstyle\sum_{\substack{j=1\\ j\neq l}}^{3}A_j^T\bar{t}_{jl}+\bar{r}_l+A_l^T\bar{s}_l=0\\ &\bar{s}_l=(\bar{s}_l+\gamma(A_l\bar{x}_l-a_l))_+ \qquad\qquad l=1,2,3,\quad j=1,2,3,\quad j\neq l,\\ &\bar{t}_{jl}=(\gamma(A_j\bar{x}_l-a_j)+\bar{p}_{jl})_+ \end{aligned}$$

where

$$\bar{r}_l=\sum_{\substack{j=1\\ j\neq l}}^{3}A_j^T(\bar{s}_j-\bar{t}_{jl}) \qquad l=1,2,3,$$

and hence that

$$\begin{aligned} &c+Q\bar{x}_l+\textstyle\sum_{j=1}^{3}A_j^T\bar{s}_j=0\\ &\bar{s}_l=(\bar{s}_l+\gamma(A_l\bar{x}_l-a_l))_+ \qquad\qquad l=1,2,3. \end{aligned} \tag{2.31}$$

It is now clear from the nonsingularity of Q that

$$\bar{x}_1 = \bar{x}_2 = \bar{x}_3 =: \bar{x}.$$

Conditions (2.31) then become the necessary and sufficient conditions for $\bar{x}$ to be the unique solution of (2.1) with multipliers as indicated in the statement of the theorem. Furthermore, since $\bar{z} = (\bar{s}_1, \bar{s}_2, \bar{s}_3, \bar{t}_{12}, \bar{t}_{23}, \bar{t}_{31}, \bar{t}_{13}, \bar{t}_{21}, \bar{t}_{32})$ solves the 3-block LCP (2.12) with identical M and q subblocks as defined by (2.23) and (2.11), respectively, it follows that each of $(\bar{s}_1, \bar{s}_2, \bar{s}_3)$, $(\bar{t}_{12}, \bar{t}_{23}, \bar{t}_{31})$, and $(\bar{t}_{13}, \bar{t}_{21}, \bar{t}_{32})$ solve any one of the three subblocks of LCP (2.12) and hence [13, Cor. 2] their differences lie in the nullspace of H. Thus

$$(2.32) \quad H\left(\begin{bmatrix} \bar{s}_1 \\ \bar{s}_2 \\ \bar{s}_3 \end{bmatrix} - \begin{bmatrix} \bar{t}_{12} \\ \bar{t}_{23} \\ \bar{t}_{31} \end{bmatrix}\right) = 0 \quad \text{and} \quad H\left(\begin{bmatrix} \bar{s}_1 \\ \bar{s}_2 \\ \bar{s}_3 \end{bmatrix} - \begin{bmatrix} \bar{t}_{13} \\ \bar{t}_{21} \\ \bar{t}_{32} \end{bmatrix}\right) = 0.$$

Relations (2.32), and relations (2.21) in the limit, imply that $\bar{p}_{jl} = \bar{t}_{jl}$, $l = 1,2,3$, $j = 1,2,3$, $j \neq l$. □

3. Parallel constraint distribution for convex programs. We extend our ideas now to general convex programs with strongly convex objective functions. For simplicity of notation we consider the 2-block problem

$$(3.1) \qquad \begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g_1(x) \leq 0, \qquad g_2(x) \leq 0, \end{aligned}$$

where $f: \mathbb{R}^n \to \mathbb{R}$, $g_1: \mathbb{R}^n \to \mathbb{R}^{m_1}$, $g_2: \mathbb{R}^n \to \mathbb{R}^{m_2}$ are differentiable convex functions on $\mathbb{R}^n$, with f strongly convex with modulus k, and g_1, g_2 Lipschitz continuous with constant K on $\mathbb{R}^n$. We begin with the following straightforward Lipschitz continuity result.

LEMMA 3.1. *Let f, g_1, g_2 be differentiable convex functions on $\mathbb{R}^n$ with f strongly convex with modulus k, and let g_1 be Lipschitz continuous with constant K on $\mathbb{R}^n$. Let g_2 satisfy a constraint qualification on the nonempty set $\{x \mid g_2(x) \leq 0\}$. Then*

$$x(u_1) := \arg\min \left\{ f(x) + \frac{1}{2\gamma} \left\{ \left\| (\gamma g_1(x) + u_1)_+ \right\|^2 \right\} \mid g_2(x) \leq 0 \right\}$$

is Lipschitz continuous on $\mathbb{R}^{m_1}_+$ with Lipschitz constant $(K/2k)(1 + \sqrt{1 + 4k/\gamma K^2})$.

Proof. Let u_1, $\bar{u}_1 \in \mathbb{R}^{m_1}_+$ and $x := x(u_1)$ and $\bar{x} := x(\bar{u}_1)$. By the Karush–Kuhn–Tucker conditions, there exist v_2, $\bar{v}_2 \in \mathbb{R}^{m_2}$ such that

$$\begin{gathered} \nabla f(x) + (\gamma g_1(x) + u_1)_+^T \nabla g_1(x) + v_2^T \nabla g_2(x) = 0, \\ g_2(x) \leq 0, \qquad \langle v_2, g_2(x) \rangle = 0, \qquad v_2 \geq 0 \end{gathered}$$

and

$$\begin{gathered} \nabla f(\bar{x}) + (\gamma g_1(\bar{x}) + \bar{u}_1)_+^T \nabla g_1(\bar{x}) + \bar{v}_2^T \nabla g_2(\bar{x}) = 0, \\ g_2(\bar{x}) \leq 0, \qquad \langle \bar{v}_2, g_2(\bar{x}) \rangle = 0, \qquad \bar{v}_2 \geq 0. \end{gathered}$$

By the strong convexity of f we have that

$$k \left\| \bar{x} - x \right\|^2 \leq (\nabla f(\bar{x}) - \nabla f(x))(\bar{x} - x).$$

This, together with the Karush–Kuhn–Tucker conditions, gives

$$\begin{aligned} k\|\bar{x}-x\|^2 &\leq \Big((\gamma g_1(\bar{x})+\bar{u}_1)_+^T \nabla g_1(\bar{x}) + \bar{v}_2^T \nabla g_2(\bar{x}) \\ &\qquad - (\gamma g_1(x)+u_1)_+^T \nabla g_1(x) - v_2^T \nabla g_2(x)\Big)(x-\bar{x}) \\ &\leq \langle (\gamma g_1(x)+u_1)_+ - (\gamma g_1(\bar{x})+\bar{u}_1)_+ , g_1(\bar{x}) - g_1(x)\rangle \\ &\quad + \langle v_2 - \bar{v}_2, g_2(\bar{x}) - g_2(x)\rangle , \end{aligned}$$

where the last inequality above follows from the following inequality:

$$\langle w-\bar{w}, h(x)-h(\bar{x})\rangle \leq (w^T\nabla h(x) - \bar{w}^T\nabla h(\bar{x}))(x-\bar{x})$$

for a convex differentiable $h\colon \mathbb{R}^n \to \mathbb{R}^k$ and $w, \bar{w} \in \mathbb{R}^k_+$. The Karush–Kuhn–Tucker conditions allow us to drop the nonpositive term $\langle v_2 - \bar{v}_2, g_2(\bar{x}) - g_2(x)\rangle$, thus giving us

$$k\|\bar{x}-x\|^2 \leq \langle (\gamma g_1(x)+u_1)_+ - (\gamma g_1(\bar{x})+\bar{u}_1)_+ , g_1(\bar{x}) - g_1(x)\rangle .$$

From the fundamental properties of the projection operator $(\ )_+$, we have for $y, z \in \mathbb{R}^m$, $\langle y-z, (y)_+ - (z)_+\rangle \geq 0$, so that

$$\begin{aligned} k\|\bar{x}-x\|^2 &\leq \frac{1}{\gamma}\langle (\gamma g_1(x)+u_1)_+ - (\gamma g_1(\bar{x})+\bar{u}_1)_+ , u_1-\bar{u}_1\rangle \\ &\leq \frac{1}{\gamma}\|(\gamma g_1(x)+u_1)_+ - (\gamma g_1(\bar{x})+\bar{u}_1)_+\| \, \|u_1-\bar{u}_1\| \\ &\leq \frac{1}{\gamma}\|\gamma g_1(x)+u_1-\gamma g_1(\bar{x})-\bar{u}_1\| \, \|u_1-\bar{u}_1\| \\ &\leq K\|x-\bar{x}\| \, \|u_1-\bar{u}_1\| + \frac{1}{\gamma}\|u_1-\bar{u}_1\|^2 . \end{aligned}$$

Defining $d := \|\bar{x}-x\|$ and $e := \|u_1-\bar{u}_1\|$ we obtain the quadratic inequality in d

$$kd^2 - Ked - \frac{1}{\gamma}e^2 \leq 0$$

and hence d must lie between the roots

$$d = \frac{Ke \pm \sqrt{K^2e^2 + 4ke^2/\gamma}}{2k}.$$

Thus $d \leq (K/2k)\,[1+\sqrt{1+4k/\gamma K^2}]e$, which gives the required Lipschitz continuity. □

We are now able to state a parallel constraint distribution algorithm for the convex program (3.1) and establish its convergence.

THEOREM 3.2 (PCD algorithm and convergence for convex programs). *Let $f\colon \mathbb{R}^n \to \mathbb{R}$, $g_1\colon \mathbb{R}^n \to \mathbb{R}^{m_1}$, $g_2\colon \mathbb{R}^n \to \mathbb{R}^{m_2}$ be continuously differentiable convex functions on $\mathbb{R}^n$ with f strongly convex and g_1, g_2 Lipschitz continuous on $\mathbb{R}^n$. Let*

$$g(x) := \begin{bmatrix} g_1(x) \\ g_2(x) \end{bmatrix}$$

and let g_1 *and* g_2 *satisfy some constraint qualification on the nonempty sets* $\{x \mid g_1(x) \le 0\}$ *and* $\{x \mid g_2(x) \le 0\}$, *respectively. Define*

$$s^i := \begin{bmatrix} s_1^i \\ s_2^i \end{bmatrix} \in \mathbb{R}^{m_1+m_2}$$

and start with $s_1^0 = 0$, $s_2^0 = 0$. *Given* s^i, *determine* s^{i+1} *as follows:*

$$\begin{aligned} (x_1^{i+1}, s_1^{i+1}) &\in \arg \mathrm{KKT} \left(\min \left\{ \mathrm{f(x)} + \frac{1}{2\gamma} \left\| \left(\gamma \mathrm{g_2(x)} + \mathrm{s_2^i}\right)_+ \right\|^2 \mid \mathrm{g_1(x)} \le 0 \right\} \right), \\ (x_2^{i+1}, s_2^{i+1}) &\in \arg \mathrm{KKT} \left(\min \left\{ \mathrm{f(x)} + \frac{1}{2\gamma} \left\| \left(\gamma \mathrm{g_1(x)} + \mathrm{s_1^i}\right)_+ \right\|^2 \mid \mathrm{g_2(x)} \le 0 \right\} \right). \end{aligned} \tag{3.2}$$

Assume that $\{s^{i+1} - s^i\} \to 0$; *then for each accumulation point* $\bar{s}$ *such that* $\{s^{i_j}\} \to \bar{s}$, $\{x_1^{i_j}\}$ *and* $\{x_2^{i_j}\}$ *converge to* $\arg\min\{f(x) \mid g(x) \le 0\}$.

Proof. By Lemma 3.1, $x_1^{i+1} := x_1(s^i)$, $x_2^{i+1} := x_2(s^i)$ are continuous. Let $\{s^{i_j}\} \to \bar{s}$. Hence $\{s^{i_j+1}\} \to \bar{s}$, $\{x_1^{i_j}\} \to \bar{x}_1$, and $\{x_2^{i_j}\} \to \bar{x}_2$. Invoking the continuity of the Karush–Kuhn–Tucker conditions, we have at these limits

$$\begin{gathered} \nabla f(\bar{x}_1) + (\gamma g_2(\bar{x}_1) + \bar{s}_2)_+^T \nabla g_2(\bar{x}_1) + \bar{s}_1^T \nabla g_1(\bar{x}_1) = 0, \\ \bar{s}_1 = (\gamma g_1(\bar{x}_1) + \bar{s}_1)_+ , \end{gathered}$$

and

$$\begin{gathered} \nabla f(\bar{x}_2) + (\gamma g_1(\bar{x}_2) + \bar{s}_1)_+^T \nabla g_1(\bar{x}_2) + \bar{s}_2^T \nabla g_2(\bar{x}_2) = 0, \\ \bar{s}_2 = (\gamma g_2(\bar{x}_2) + \bar{s}_2)_+ . \end{gathered}$$

Hence

$$\nabla f(\bar{x}_1) + (\gamma g_2(\bar{x}_1) + \bar{s}_2)_+^T \nabla g_2(\bar{x}_1) + (\gamma g_1(\bar{x}_1) + \bar{s}_1)_+^T \nabla g_1(\bar{x}_1) = 0$$

and

$$\nabla f(\bar{x}_2) + (\gamma g_1(\bar{x}_2) + \bar{s}_1)_+^T \nabla g_1(\bar{x}_2) + (\gamma g_2(\bar{x}_2) + \bar{s}_2)_+^T \nabla g_2(\bar{x}_2) = 0.$$

Thus

$$\bar{x}_1 = \bar{x}_2 = \arg\min \left\{ f(x) + \frac{1}{2\gamma} \left\| (\gamma g(x) + \bar{s})_+ \right\|^2 \right\},$$

because the objective of the last minimization problem is strongly convex. Hence $(\bar{x}_1, \bar{s})$ and $(\bar{x}_2, \bar{s})$ satisfy the Karush–Kuhn–Tucker conditions of $\min\{f(x) \mid g(x) \le 0\}$ and thus $\bar{x}_1 = \bar{x}_2 = \arg\min\{f(x) \mid g(x) \le 0\}$. □

Without going into much detail, we note that it is possible under suitable assumptions to solve for x_1^{i+1} in terms of (s_1^{i+1}, s_2^i) and x_2^{i+1} in terms of (s_1^i, s_2^{i+1}), in which case the PCD algorithm (Theorem 3.2) can be rewritten as the following nonlinear Jacobi iteration [18] for solving a nonlinear complementarity problem:

$$\begin{aligned} s_1^{i+1} &= \left(\gamma g_1(x_1(s_1^{i+1}, s_2^i)) + s_1^{i+1}\right)_+ , \\ s_2^{i+1} &= \left(\gamma g_2(x_2(s_1^i, s_2^{i+1})) + s_2^{i+1}\right)_+ . \end{aligned} \tag{3.3}$$

Improved convergence proofs (see, for example, [18]) may be possible, based on this equivalent Jacobi iteration instead of (3.2).

4. Computational experience. We have tested out the algorithms of the previous sections on some linear programming problems. The standard form linear program

$$\begin{aligned} &\text{minimize} && c^T x \\ &\text{subject to} && Ax = b, \qquad x \geq 0 \end{aligned}$$

has the dual problem

$$\begin{aligned} &\text{maximize} && b^T y \\ &\text{subject to} && A^T y \leq c \end{aligned} \tag{4.1}$$

and these problems are in precisely the form of our preceding discussion except that the objective is not strongly convex. In order to strongly convexify the objective we have used the least 2-norm formulation [15], [12], where for $\epsilon \in (0, \bar{\epsilon}]$ for some $\bar{\epsilon} > 0$, the solution of

$$\begin{aligned} &\text{minimize} && -b^T y + \tfrac{\epsilon}{2} y^T y \\ &\text{subject to} && A^T y \leq c \end{aligned} \tag{4.2}$$

is the least 2-norm solution of (4.1). For the purpose of our computation, a value of $\epsilon = 10^{-6}$ was used.

We have split up the problems as follows: first, the user has specified the number of processors available, and the problem has been split into that many blocks. If the number of constraints in each block is not the same, we have added to each block combinations of constraints from other blocks to make the number of constraints in each block equal, with the aim of balancing the load between processors.

The PCD algorithm (Theorem 3.2) of §3 was implemented on the Sequent Symmetry S-81 shared memory multiprocessor. The subproblems were solved on each processor using MINOS 5.3, a more recent version of [16]. The explicit constraints in each subproblem remained fixed throughout the computation, but the blocks were not chosen to satisfy the linear independence assumption.

We have used the following heuristic scheme to update the augmented Lagrangian parameter γ. Initially it is set at 10 and is increased by a factor of 4 only when the norm of the violation of the constraints increases.

The steplength λ in the method (which is needed in the convergence proof) was chosen by several techniques. One technique was to choose a fixed positive steplength $\lambda < 1$. With a steplength of 1 we found that the algorithm did fail to converge in several instances, as the theory would suggest (see Table 4.2). We have also experimented with a heuristic choice of the steplength λ. We calculated a merit function at certain values of λ between 0.4 and 1.0 (depending on the number of processors available) and took the step λ from among these values, which minimized the merit function. The particular form of merit function we employ is a weighted sum of two quantities, the first being the norm of the gradient of the standard Lagrangian for (4.2) and the second being the difference between the objective function values of (4.2) and its dual. This has proven to be robust and results in a good saving in iterations (see Table 4.3). Also, the evaluation of the merit function was extremely cheap to perform (in parallel) and did not result in any degrading of the parallel performance.

The algorithm was terminated whenever the difference in the primal objective value of (4.1) and its dual objective value normalized by their sum differed by less than 10^{-5}. The constraint violation was also required to be less than this tolerance.

TABLE 4.1
Numerical results with fixed $\lambda = 0.7$.

Problem	No. of Variables	No. of Constraints	Iteration Count for No. of Blocks			
			3	6	9	18
Ex6	3	5	10	10		
Ex9	5	11	10	12		
Ex10	6	14	11	12	13	
AFIRO	27	51	16	16	16	16
ADLittle	56	138	14	27	19	27

TABLE 4.2
Numerical results with fixed $\lambda = 1.0$.

Problem	No. of Variables	No. of Constraints	Iteration Count for No. of Blocks			
			3	6	9	18
Ex6	3	5	2	2		
Ex9	5	11	4	*		
Ex10	6	14	4	4	4	
AFIRO	27	51	20	14	*	*
ADLittle	56	138	*	*	*	*

TABLE 4.3
Numerical results with variable λ.

Problem	No. of Variables	No. of Constraints	Iteration Count for No. of Blocks			
			3	6	9	18
Ex6	3	5	2	2		
Ex9	5	11	4	5		
Ex10	6	14	4	4	4	
AFIRO	27	51	13	15	15	14
ADLittle	56	138	12	14	14	15

Tables 4.1, 4.2, and 4.3 summarize preliminary numerical results for the PCD algorithm (Theorem 3.2) on the Sequent Symmetry S-81 for five small linear programs reformulated as in (4.2). The first three are homemade test problems, while the last two, AFIRO and ADLittle, are from the NETLIB collection [7]. In the tables, an empty column entry signifies that we did not perform the computation. The character * signifies that the algorithm did not terminate. Note that the algorithm does fail when a full step is taken (see Table 4.2), as may be expected from Theorem 2.1, where the stepsize λ must satisfy (2.30). The heuristic stepsize outlined above performs the best (see Table 4.3).

The key observation to make is that the total number of iterations required for accurate solutions (tolerance less than 10^{-5}) can be achieved with a small number of iterations (2–13 iterations for 3 blocks and 14–15 iterations for 18 blocks). The fact that the number of iterations remains essentially constant for increasing number of blocks is encouraging and leads us to believe that the PCD is worthy of additional theoretical and computational study.

REFERENCES

[1] D. BERTSEKAS, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, 1982.

[2] R. COTTLE, J.-S. PANG, AND R. STONE, *The Linear Complementarity Problem*, Academic Press, New York, 1991.

[3] R. DE LEONE AND O. MANGASARIAN, *Parallel proximal point decomposition of linear programming constraints*, SIAM National Meeting, Chicago, IL, July 16–20, 1990.

[4] A. DE PIERRO AND A. IUSEM, *Convergence properties of iterative methods for symmetric positive semidefinite linear complementarity problems*, Tech. Report, Instituto de Matematica, Elasticita e Ciencia da Computacao, Universidade Estadual de Campinas, Campinas, Brazil, 1990.

[5] J. ECKSTEIN AND D. BERTSEKAS, *On the Douglas–Rachford splitting method and the proximal point algorithm for maximal monotone operators*, Math. Programming, 1991, to appear.

[6] M. FERRIS, *Parallel constraint distribution for convex quadratic programs*, Tech. Report 1009, Computer Sciences Department, University of Wisconsin, Madison, WI, 1991.

[7] D. GAY, *Electronic mail distribution of linear programming test problems*, COAL Newsletter, 13 (1985), pp. 10–12.

[8] S.-P. HAN, *A decomposition method and its application to convex programming*, Math. Oper. Res., 14 (1989), pp. 237–248.

[9] Z.-Q. LUO AND P. TSENG, *On the convergence of a matrix splitting algorithm for the symmetric monotone linear complementarity problem*, Tech. Report LIDS-P-1884, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1989; SIAM J. Control Optim., 30 (1992), to appear.

[10] O. MANGASARIAN, *Nonlinear Programming*, McGraw–Hill, New York, 1969.

[11] ———, *Solution of symmetric linear complementarity problems by iterative methods*, J. Optim. Theory and Appl., 22 (1977), pp. 465–485.

[12] ———, *Normal solutions of linear programs*, Math. Programming Stud., 22 (1984), pp. 206–216.

[13] ———, *A simple characterization of solution sets of convex programs*, Oper. Res. Lett., 7 (1988), pp. 21–26.

[14] ———, *Convergence of iterates of an inexact matrix splitting algorithm for the symmetric monotone linear complementarity problem*, SIAM J. Optimization, 1 (1991), pp. 114–122.

[15] O. MANGASARIAN AND R. MEYER, *Nonlinear perturbation of linear programs*, SIAM J. Control Optim., 17 (1979), pp. 745–752.

[16] B. MURTAGH AND M. SAUNDERS, MINOS 5.0 *user's guide*, Tech. Report SOL 83.20, Systems Optimization Laboratory, Stanford University, December 1983.

[17] J.-S. PANG, *Convergence of splitting and Newton methods for complementarity problems: An application of some sensitivity results*, Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, MD, September, 1990.

[18] J.-S. PANG AND D. CHAN, *Iterative methods for variational and complementarity problems*, Math. Programming, 24 (1982), pp. 284–313.

[19] R. ROCKAFELLAR, *Augmented Lagrange multiplier functions and duality in nonconvex programming*, SIAM J. Control, 12 (1974), pp. 268–285.

[20] ———, *Augmented Lagrangians and applications of the proximal point algorithm in convex programming*, Math. Oper. Res., 1 (1976), pp. 97–116.

[21] R. ROCKAFELLAR AND R.-B. WETS, *Scenarios and policy aggregation in optimization under uncertainty*, Math. Oper. Res., 10 (1991), pp. 119–147.

[22] J. SPINGARN, *Applications of the method of partial inverses to convex programming*, Math. Programming, 32 (1985), pp. 199–223.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 501–514, November 1991

PARALLEL SOLUTION OF LARGE-SCALE, BLOCK-DIAGONAL CONCAVE MAXIMIZATION PROBLEMS*

J. H. GLICK†, R. S. MAIER†‡, AND J. B. ROSEN†

Abstract. A feasible-point algorithm for structured, large-scale, constrained optimization problems which may have many nonlinear constraints is described. The constraint structure is characterized by a block-diagonal coefficient matrix corresponding to linear variables, coupled by nonlinear variables. Problems of this structure arise in many applications, including structural design optimization and certain multiperiod or multiplant applications. Maximization problems with a concave objective and concave inequality constraints which define a convex region are considered. For such problems a KKT point is a global maximum. A basic version of the algorithm is presented and justified by showing that it will find an optimal solution in a finite number of iterations. The algorithm has been implemented on a CRAY-2 and a 64-processor NCUBE hypercube. It has been tested on a series of randomly generated test problems, and its performance has been compared with that of MINOS 5.3.

Key words. nonlinear programming, large-scale problems, constrained optimization

AMS(MOS) subject classifications. 90C30, 90C06

1. Introduction.

Many large-scale, constrained, nonlinear programming problems have certain common features; the constraint matrix is structured and sparse, and many of the variables are linear. Problems of this general type occur in many important large-scale optimization applications, including combined manufacturing and distribution models [2] and structural design optimization [3]. Such problems may also have many nonlinear constraints.

In this paper we limit consideration to the case where the feasible set is convex and the objective function is concave for a maximization (or convex for a minimization) problem. That is, we consider problems where the first-order optimality conditions (KKT conditions) are sufficient for a global optimum. An efficient method for the solution of this type of problem is presented here, including computational tests with sequential and parallel implementations.

The block-diagonal concave problem considered is given by:

$$\max_{y,w}\{f(y) + b^T w \mid A^T w \leq c(y)\}, \tag{1}$$

where $y \in R^s$, $w, b \in R^m$, $A \in R^{m \times n}$ is block-diagonal with k blocks, $f : R^s \to R$, $c : R^s \to R^n$, and f, c_i, $i = 1, \cdots, n$ are concave, twice differentiable functions. It is assumed that $\text{rank}(A) = m$. The number s of linking variables (the vector y) is typically much smaller than the number m of linear variables. The total number n of inequality constraints is assumed greater than m (these may include simple bounds and linear inequalities). It should be noted that the feasible set defined by $A^T w \leq c(y)$ is convex. To simplify the presentation it is assumed that a KKT point $(y^\star, w^\star)$ exists (i.e., the problem has an optimal solution), and that $(y^\star, w^\star)$ is nondegenerate, in that at most $m + s$ of the n inequalities are satisfied as equalities. With the block-diagonal

* Received by the editors October 3, 1990; accepted for publication (in revised form) April 3, 1991. This work was supported in part by Air Force Office of Scientific Research grant AFOSR-87-0127, the Minnesota Supercomputer Institute, and the Army High Performance Computing Research Center.

† Department of Computer Science, University of Minnesota, Minneapolis, Minnesota 55455.

‡ Army High Performance Computing Research Center, University of Minnesota, Minneapolis, Minnesota 55415.

structure assumed for A, the problem constraints can be written:

$$\begin{pmatrix} A^1 & & \\ & \ddots & \\ & & A^k \end{pmatrix}^T \begin{pmatrix} w^1 \\ \vdots \\ w^k \end{pmatrix} \begin{matrix} \leq \\ \vdots \\ \leq \end{matrix} \begin{pmatrix} c^1(y) \\ \vdots \\ c^k(y) \end{pmatrix}. \tag{2}$$

Here, vectors and matrices associated with individual blocks are denoted by superscripts.

A property of this class of structured problems is that the constraints decouple into separate blocks for any fixed value of the nonlinear variables. The method presented takes full advantage of this property. For a survey of this class of problems, see [3].

For the special case where both $f(y)$ and $c(y)$ are affine functions, that is, $f(y) = b_0^T y$ and $c(y) = c - D^T y$, where $D \in R^{s \times n}$, the problem is a sparse linear program with a dual block-angular structure containing s coupling columns. Considering this as the dual problem, the equivalent primal is

$$\min_x \{c^T x \mid Dx = b_0,\ Ax = b,\ x \geq 0\}. \tag{3}$$

An efficient parallel method for solving this large-scale structured linear program has previously been described, implemented, and computationally tested in [9] and [5].

The original problem (1) can be represented in the reduced space of the linking variables $y \in R^s$, by defining the function

$$\Psi(y) = f(y) + \max_w \{b^T w \mid A^T w \leq c(y)\}. \tag{4}$$

Let $Y \subset R^s$ be the set of y such that there exists a $w \in R^m$ for which the inequalities $A^T w \leq c(y)$ are satisfied. The set Y is not empty since $y^\star \in Y$ and it is straightforward to show that it is convex. Furthermore, $\Psi(y)$ is a concave, piecewise differentiable function of y defined over the set Y. Its value for any fixed $y \in Y$ is easily computed by solving k independent, and relatively small, linear programs

$$\max_{w^i} \{(b^i)^T w^i \mid (A^i)^T w^i \leq c^i(y)\}, \qquad i = 1, \cdots, k, \tag{5}$$

which can all be done in parallel (with k processors). The problem (1) can then be stated as a constrained maximization in y-space:

$$\max_{y \in Y} \Psi(y). \tag{6}$$

If we consider the linear program in (4) as a dual problem, the equivalent primal problem (for any fixed y) is given by

$$\min_u \{c(y)^T u \mid Au = b,\ u \geq 0\}. \tag{7}$$

This problem is referred to as the LP subproblem in the remainder of the paper. In terms of the block-diagonal structure this gives the k independent primal linear programs

$$\min_{u^i} \{c^i(y)^T u^i \mid A^i u^i = b^i,\ u^i \geq 0\}, \qquad i = 1, \cdots, k. \tag{8}$$

The formulation (6) shows that the large-scale original problem (1) can be reduced to a much smaller problem in R^s. However, the formulation (6) has two serious

difficulties: $\Psi(y)$ is only piecewise differentiable and the feasible set Y is not defined explicitly. To avoid these difficulties, the formulation is replaced by a sequence of reduced problems, each of which corresponds to a fixed basis B selected from the columns of A. The reduced problem is given by

$$\max_y \{\Phi(y) \mid \pi_i(y) \geq 0,\ i \text{ nonbasic}\}, \tag{9}$$

where $\Phi(y)$ and $\pi_i(y)$ can be explicitly computed using B^{-1}, and are differentiable functions for all $y \in R^s$. The reduced problem is based on ideas presented in [8], and is described in detail in the next section.

Starting with any $y_0 \in Y$, a sequence of reduced problems is solved, giving a sequence of strictly increasing function values. The first-order (KKT) optimality conditions for the reduced problem are closely related to the KKT conditions for (1). In fact, satisfaction of the reduced problem KKT conditions, together with an additional nonnegativity requirement on the basic multipliers, implies that the KKT conditions for (1) are satisfied, and therefore an optimum solution to (1) has been obtained. If the additional nonnegativity requirement is not satisfied, a different basis is chosen and a new reduced problem formed and solved. This iteration is continued until the optimal solution is obtained in a finite number of steps. Details of this RMG (Rosen–Maier–Glick) algorithm are given in §§3 and 4.

In §2 the KKT conditions for (1) are given. It is then shown that, corresponding to any feasible basis B, a reduced problem can be generated. The KKT conditions for the reduced problem are given and their relation to the KKT conditions for (1) shown. An exchange of columns is needed in the RMG algorithm. It is shown that this can be done whenever it is required.

In §3 a basic version of the RMG algorithm is presented and justified. It is shown that the optimum is obtained in a finite number of iterations. An heuristic modification is described in §4, which has proved very efficient in practice.

In §5 the generation of problems for the computational testing and comparison of the RMG algorithm is described. To determine the time dependence on problem size, problems with from 1 to 64 blocks were solved, with block size held fixed, and m constraints active at the optimum. The RMG implementation was tested on both a Cray-2 and a first-generation, 64-processor NCUBE. The largest problem solved, with 3200 variables and 6400 quadratic constraints, required 91 seconds on the Cray-2. A comparison was also made with MINOS 5.3 using the same test problems. For a discussion of MINOS, see [6] and [7].

2. Optimality conditions and the reduced problem. The first-order (KKT) optimality conditions for the problem (1) are given by:

$$\begin{aligned} c(y) - A^T w &\geq 0, \\ \nabla f(y) + D(y)x &= 0, \\ Ax &= b, \\ x &\geq 0, \\ x^T(c(y) - A^T w) &= 0, \end{aligned} \tag{10}$$

where $D(y) \in R^{s \times n}$ has columns $\nabla c_i(y)$, $i = 1, \cdots, n$, and $x \in R^n$ are the Lagrange multipliers. Since $f(y)$ and the components of $c(y)$ are concave functions, the conditions (10) are sufficient for a global optimum to (1). We also assume appropriate

constraint qualifications, so that these conditions are necessary. By our earlier assumption that an optimum solution $(y^\star, w^\star)$ exists, there exists $x^\star \geq 0$, such that $(y^\star, w^\star, x^\star)$ satisfy (10).

The reduced problem is obtained by using an $m \times m$ nonsingular basis B, chosen from the columns of A, so that A is partitioned:

$$A = [B\ N].$$

Corresponding to this partition, let $I_B = \{i \mid a_i \text{ is a column of } B\}$ and $I_N = \{i \mid a_i$ is a column of $N\}$. When convenient, subscripting with B or N on vectors refers to components corresponding to I_B and I_N, respectively.

The choice of B must be such that for some (y, w) we have $B^T w = c_B(y)$ and $N^T w \leq c_N(y)$.

Usually this is accomplished by solving the linear program (7) with a specific $y = \bar{y}$, to give B, N, and $u_B = B^{-1}b \geq 0$. This $\bar{y}$ can be the initial value or the value from a previous algorithm iterate. The vector $c(\bar{y}) - A^T w \geq 0$ is then the optimal reduced cost vector for (7), where $w = B^{-T} c_B(\bar{y})$. For any such basis B, we eliminate w by the relation

$$w = B^{-T} c_B(y)$$

and use this in (1) to get

$$\begin{aligned} \Phi(y) &= f(y) + c_B^T(y) B^{-1} b, \\ \pi_i(y) &= c_i(y) - c_B^T(y) B^{-1} a_i, \qquad i \in I_N. \end{aligned}$$

Note that for fixed B, both Φ and π_i are differentiable for all $y \in R^s$. The reduced problem corresponding to B is

$$\max_y \{\Phi(y) \mid \pi_i(y) \geq 0,\ i \in I_N\}. \tag{11}$$

The vector $\bar{y}$ is feasible for (11), because the $\pi_i(\bar{y})$ represent the nonbasic reduced costs corresponding to the optimal basis B, and therefore $\pi_i(\bar{y}) \geq 0$, $i \in I_N$. Furthermore, any $y \in R^s$ which is feasible for (11) is also feasible for (1), and the function value $\Phi(y)$ is equal to the function value of (1) corresponding to the basis B. Clearly, the problem (11) has fewer variables and constraints than (1), since (11) has only s variables and $n - m$ constraints. Typically, s will be much smaller than m. Furthermore, by the nondegeneracy assumption, the number of active constraints in the reduced problem at optimality will be $q \leq s$. For later use we also define the reduced gradient

$$g(y) = \nabla \Phi(y) = \nabla f(y) + D_B(y) B^{-1} b.$$

Corresponding to any feasible y, we define a subset $I_S \subset I_N$, of active constraints, such that

$$I_S = \{i \mid \pi_i(y) = 0,\ i \in I_N\}. \tag{12}$$

For all other constraints we have $\pi_i(y) > 0$, for $i \in I_N \setminus I_S$. Multipliers x_i, for $i \in I_S$, are called superbasic. As with B and N, subscripting with S on a vector refers to components corresponding to I_S.

The KKT conditions for (11) at $y^\star$ with multipliers $x_N^\star$ can then be given as follows:

$$
\begin{aligned}
\pi_i(y^\star) &\geq 0, \qquad i \in I_N, \\
\pi_i(y^\star) &= 0, \qquad i \in I_S, \\
\nabla\Phi(y^\star) + \sum_{i \in I_S} x_i^\star \nabla\pi_i(y^\star) &= 0, \\
x_i^\star &\geq 0, \qquad i \in I_S, \\
x_i^\star &= 0, \qquad i \in I_N \setminus I_S.
\end{aligned} \tag{13}
$$

The reduced problem (11) and the original problem (1) are obviously closely related. We now show that with one additional condition, the satisfaction of (13) is sufficient for $y^\star$ and $w^\star = B^{-T}c_B(y^\star)$ to be the optimal solution to (1).

THEOREM 2.1. *Let $y^\star$ and $x_i^\star \geq 0$, $i \in I_N$ satisfy* (13). *Also let $u_B = B^{-1}b$ and*

$$
x_B^\star = u_B - B^{-1}\left(\sum_{i \in I_S} x_i^\star a_i\right) \geq 0. \tag{14}
$$

Then $(y^\star, w^\star)$ is an optimal solution to (1).

Proof. From (13), (14), and the definitions of $\Phi(y)$ and $\pi(y)$, we have

$$
\begin{aligned}
0 &= \nabla\Phi(y^\star) + \sum_{i \in I_S} x_i^\star \nabla\pi_i(y^\star) \\
&= \nabla f(y^\star) + D_B(y^\star)u_B \\
&\quad + \sum_{i \in I_S} x_i^\star \nabla c_i(y^\star) - \sum_{i \in I_S} x_i^\star D_B(y^\star)B^{-1}a_i \\
&= \nabla f(y^\star) + D_B(y^\star)x_B^\star + \sum_{i \in I_S} x_i^\star \nabla c_i(y^\star) \\
&= \nabla f(y^\star) + \sum_{i \in I_S \cup I_B} x_i^\star \nabla c_i(y^\star).
\end{aligned}
$$

Thus, the condition $\nabla f(y^\star) + D(y^\star)x^\star = 0$ of (10) is satisfied. The feasibility requirement $c(y^\star) - A^T w^\star \geq 0$ is satisfied since $c_B(y^\star) - B^T w^\star = 0$, and $\pi_i(y^\star) = c_i(y^\star) - a_i^T w^\star \geq 0$, $i \in I_N$. Also,

$$
Ax^\star = Bx_B^\star + \sum_{i \in I_N} x_i^\star a_i = b, \qquad x^\star \geq 0 \tag{15}
$$

is satisfied by (14) and the requirement that $x_i^\star \geq 0$, $i \in I_N$. Finally, the complementarity condition ${x^\star}^T(c(y^\star) - A^T w^\star) = 0$ is satisfied since $c_i(y^\star) - a_i^T w^\star = 0$ for $i \in I_B \cup I_S$, and $x_i^\star = 0$ for $i \in I_N \setminus I_S$. All the KKT conditions (10) are satisfied, and since they are sufficient conditions, $(y^\star, w^\star)$ is optimal for (1). □

The following lemma is needed in the next section.

LEMMA 2.2 (COLUMN EXCHANGE). *If $u_B \geq 0$ and the optimal solution to the reduced problem causes at least one of the basic multipliers to be negative, an exchange of a nonbasic column, corresponding to a positive superbasic multiplier, with a basic column, can always be made.*

Proof. We have

$$x_B = u_B - \sum_{i \in I_S} x_i B^{-1} a_i \quad \text{and} \quad x_i \geq 0 \quad \text{for } i \in I_S. \tag{16}$$

Suppose $x_{B_j} < 0$. Since $u_B \geq 0$, it must be true that for at least one superbasic column (say, a_p), both $x_p > 0$ and the jth element of $B^{-1}a_p$ is positive:

$$B^{-1}a_p = \bar{a}_p = \begin{pmatrix} \bar{a}_{1p} \\ \bar{a}_{2p} \\ \vdots \\ \bar{a}_{mp} \end{pmatrix} \quad \text{and} \quad \bar{a}_{jp} > 0. \tag{17}$$

Now consider the exchange of columns in the basis. Replace a_j in the basis with the superbasic column a_p. Represent a_p in terms of basis columns:

$$a_p = \sum_{i \in I_B} \bar{a}_{ip} a_i,$$

where $\bar{a}_p$ is given by (17). To get a new basis we solve this for a_j in terms of a_p and a_i, $i \in I_B$, $i \neq j$:

$$a_j = \frac{1}{\bar{a}_{jp}} a_p - \sum_{i \in I_B, i \neq j} \frac{\bar{a}_{ip}}{\bar{a}_{jp}} a_i. \tag{18}$$

Since the pivot element is $\bar{a}_{jp} > 0$, this can always be done. A new basis B' has been obtained with columns a_p, a_i, $i \in I_B$, $i \neq j$. □

3. The Basic RMG Algorithm and convergence proof. The Basic RMG Algorithm and the proof of its finite convergence are now presented. To simplify the notation, let the function value of the reduced problem with basis B_k, be given by $\Phi_k(y) = f(y) + c_B^T(y) B_k^{-1} b$. Also, let $Y_k \subset Y$ be the feasible set for the reduced problem with basis B_k. It is assumed that an initial feasible $y_0 \in Y$ is known.

BASIC RMG ALGORITHM.

1. Set the initial value of y to $y_0 \in Y$. Solve (7) to get an initial basis B_0. Initialize the iteration count, $k \leftarrow 0$.
2. Given a basis B_k, such that $B_k^{-1} b \geq 0$, and $y_k \in Y_k$, form the reduced problem (11) with basis B_k. Compute the optimal solution to the reduced problem, starting with y_k. The optimal solution gives $y'_k \in Y_k$, the superbasic multipliers $x'_s \geq 0$, and the optimal function value $\Phi_k(y'_k)$ of the reduced problem.
3. Compute x_B as given by (16). If $x_B \geq 0$, then a solution has been found. In this case, set $y^\star \leftarrow y'_k$, $w^\star \leftarrow B_k^{-T} c_B(y'_k)$, $\Psi^\star \leftarrow \Phi_k(y'_k)$, and stop.
4. Solve (7) and use (4) to get $\Psi(y'_k)$ and the corresponding basis B'_k. If $\Psi(y'_k) > \Phi_k(y'_k)$, then it must be that $B'_k \neq B_k$. In this case, set $B_{k+1} \leftarrow B'_k$, $y_{k+1} \leftarrow y'_k$, $k \leftarrow k+1$, and go to step 2.
5. It must be that $\Psi(y'_k) = \Phi_k(y'_k)$ and at least one basic multiplier is negative. Make a basis exchange as given in the exchange lemma (Lemma 2.2) to get a new basis B_{k+1}. Form the reduced problem (11) with the basis B_{k+1}, and $\Phi_{k+1}(y'_k) = \Phi_k(y'_k)$. Find any approximate solution $y_{k+1} \in Y_{k+1}$, such that $\Phi_{k+1}(y_{k+1}) > \Phi_k(y'_k)$. Solve (7) to get $\Psi(y_{k+1}) \geq \Phi_{k+1}(y_{k+1})$, and the corresponding basis $B' \neq B_k$. Set $B_{k+1} \leftarrow B'$, $k \leftarrow k+1$, and go to step 2.

THEOREM 3.1. *Starting with any feasible y_0, the* RMG *algorithm will find the optimal solution $(y^\star, w^\star)$ to* (1) *in a finite number of iterations.*

Proof. The algorithm generates a sequence of reduced problems, each corresponding to a different basis B_k, and computes the optimal solution to each. This gives a sequence of feasible points $y_k \in Y_k \subset Y$, with increasing objective function values. Since a return to a previous basis would require a decrease in function value, no basis can be repeated. A bound on the number of different bases is $\binom{n}{m}$, so this is a bound on the number of possible iterations. A detailed proof is based on the following points:

1. The optimality test (in step 3) follows directly from Theorem 2.1 and the fact that y'_k, x'_s satisfy the KKT conditions for the reduced problem.
2. In step 4, the strict increase $\Psi(y'_k) > \Phi_k(y'_k)$ shows that a new basis has been obtained, since $\Phi_k(y'_k)$ is the maximum value possible with the basis B_k.
3. The solution of the LP subproblem (7) for any $y_k \in Y$, gives a basis B_k, such that $B_k^{-1}b \geq 0$, and corresponding function value $\Psi(y_k) = \Phi_k(y_k)$. Furthermore, $y_k \in Y_k$, since $\pi_N(y_k)$ is the optimal reduced cost vector for (7) and therefore $\pi_N(y_k) \geq 0$. In general, it should be noted that $\Psi(y) \geq \Phi_k(y)$ for any basis B_k, and $y \in Y$, since (7) chooses the optimal basis corresponding to y.
4. In step 5, the exchange can always be carried out as shown in the exchange lemma (Lemma 2.2). Since the exchange involves two active constraints in (1) at y'_k and the corresponding w, the value of w is unchanged with the new basis B_{k+1}, so that $\Phi_{k+1}(y'_k) = \Phi_k(y'_k)$. Furthermore, $y'_k \in Y_{k+1}$, that is, y'_k is feasible for the new reduced problem corresponding to B_{k+1}.
   Because of the exchange, exactly one superbasic multiplier will be negative in the new reduced problem. Therefore, by the nondegeneracy assumption, a finite step (away from the active constraint with negative multiplier) can be taken to a new feasible point y_{k+1} with increased function value (see, for example, [4, §11.4]). The LP subproblem (7) for y_{k+1} will give B', and $\Psi(y_{k+1}) \geq \Phi_{k+1}(y_{k+1}) > \Phi_{k+1}(y'_k) = \Phi_k(y'_k)$, so that $B' \neq B_k$. □

4. Computational RMG algorithm. The RMG algorithm presented above is not computationally efficient because it requires that an optimal solution to the reduced problem (11) be obtained at each iteration. To avoid this, the computational version of the algorithm requires only that if the KKT conditions are not satisfied at y_k, then a new feasible y_{k+1} is found with $\Psi(y_{k+1}) > \Psi(y_k)$. This is done by determining a feasible ascent direction d, and then computing $y_{k+1} = y_k + \alpha d$, where the scalar $\alpha > 0$ is determined by a suitable line search.

The simplest choice for d is the reduced gradient $g(y_k) = \nabla\Phi(y_k)$. This choice is best initially when y_k is not close to $y^\star$. As y_k converges to $y^\star$, $d = g(y_k)$ will tend to give a small (or possibly no) improvement in $\Psi(y)$, and therefore is not a good choice. In this case the direction d is determined by an approximate solution to the KKT conditions (13), where the set of active constraints I_S is specified. A single step of Newton's method is taken, where $\pi_i(y) = 0$, $i \in I_S$, and $g(y) + \sum_{i \in I_S} x_i \nabla\pi_i(y) = 0$ are linearized about $y = y_k$. This gives a new value $y = y'_k$, with $d = y'_k - y_k$. The permitted change in y (that is, $\|d\|$), may also be limited by a trust region.

Once d has been chosen, the step length α is selected so as to approximately solve the line search problem:

$$\max_{\alpha} \theta(\alpha), \quad \text{where } \theta(\alpha) = f(y_k + \alpha d) + \min_{x}\{c(y_k + \alpha d)^T x \mid Ax = b,\ x \geq 0\}. \tag{19}$$

Note that for each α, the LP subproblem (7) is solved to evaluate $\theta(\alpha)$. Details about

the termination tolerance used are given in §6.

5. Test problems. A class of model problems was devised to aid preliminary testing of the algorithm. These test problems are given by (1) with $f(y) = b_0^T y$, and each $c_i(y)$ a concave, quadratic function. The procedure for generating test problems begins by specifying the solution and multiplier vectors. The appropriate choice of the multipliers determines the number of superbasic variables at the solution.

PROBLEM GENERATOR.

1. Choose $y \in R^s$ (e.g., we used $y_i \in \{1, -1\}$).
2. Choose $w \in R^m$ (e.g., we used $w_i \in \{1, -1\}$).
3. Choose $A \in R^{m \times n}$ (e.g., we used integers in $\{-9, \cdots, 9\}$). Identify a set of basic columns (e.g., we used the first m columns, with a check for singularity).
4. Choose $x \in R^n$ so that $x_i > 0$ for the $m + q$ basic and superbasic columns of A $(0 \leq q \leq s)$, and $x_i = 0$ for the remaining nonbasic columns. For example, we used $x_i = 1$, $i = 1, \cdots, m + q$.
5. Compute $b = Ax$.
6. Compute $c = A^T w$.
7. Set $c_i = c_i + \alpha$ (e.g., we used $\alpha = 1$) for i corresponding to the nonbasic columns of A (the optimal prices will be α).
8. Choose a nonnegative $D \in R^{s \times n}$ (e.g., we used integers in $\{0, \cdots, 9\}$).
9. Define the vector function $c(y) = d_0 - \frac{1}{2} D^T Y^2 e$, where $Y = \text{diag}(y)$ and $e \in R^s$ is the vector of all ones. Compute $d_0 = c + \frac{1}{2} D^T Y^2 e$.
10. Compute $b_0 = YDx$.
11. Compute the optimal function value $\Psi = b_0^T y + b^T w$.

The suggested parameter choices in the problem generator satisfy necessary conditions for a maximum, and are sufficient for a global maximum since each $c_i(y)$ will be concave. In general, the parameters permit control over a number of problem characteristics, including the conditioning of the constraint functions, the structure of A, the magnitude of Lagrange multipliers, and the number of active constraints at the solution.

For the problems presented in this paper, we have deliberately chosen characteristics which make the problem relatively "easy" for the RMG algorithm (and possibly for other algorithms). The choice of D and A suggested in the generator ensure that the constraint functions are well conditioned. The point $y_0 = 0$ is a feasible starting point. Finally we have actually presented only problems with m active constraints at the solution; i.e., no superbasics at the optimal solution. This last choice is significant since such problems tend to be easier for the active set strategy currently used to solve (11). Although superbasics typically become active during the solution process on these problems, they usually pose little combinatorial difficulty in constraint identification near the optimal solution.

6. Computational results. In this section we present the results of a computational experiment comparing serial and parallel RMG codes with MINOS 5.3. Test problem sets were developed to experimentally characterize the effect of increasing the number of linear variables (the dimension of A), while the number of nonlinear variables is fixed. MINOS 5.3 is a well-known code for general purpose optimization. We believe it to be the best available code when the number of nonlinear variables is small compared to the number of linear variables and when the problems are characterized by relatively mild nonlinearities.

We emphasize our interest in structured constraints, i.e., block-diagonal A. This is the primary motivation for the RMG algorithm, and in this sense, the comparison with MINOS is simply a comparison of an algorithm which treats structure with an algorithm that does not. The appropriate way to view the comparison is that we will compare the best available general method for problems with a limited number of nonlinear variables (MINOS) with a method which exploits special structure in the linear part of the constraints (RMG). At the present time, computational testing shows that MINOS is more efficient in the unstructured case.

6.1. The Serial code. A Fortran code for the RMG algorithm has been implemented on the CRAY-2 at the Minnesota Supercomputer Center in single precision arithmetic, using CRAY-optimized BLAS routines. Constraint function and gradient evaluations are accomplished with calls to a FUNCON routine compatible with MINOS 5.3. The KKT iteration is accomplished with a call to a nonoptimized version of MINPACK's HYBRJ routine for nonlinear systems of equations. The LP is solved by a sparse routine adapted from MINOS 5.0. All RMG runs were made using one processor of the CRAY-2. Time measurements for RMG reflect the time for solving the problem and do not include setup time. The starting point for all problems was $y_0 = 0$. The termination criteria for RMG include

$$\|g\|_2/\sigma \le \epsilon^{1/3}, \qquad x \ge -\epsilon^{1/3}, \qquad \sum_{i=1}^{n} |x_i(c_i - a_i^T w)| \le 10^{-6}, \tag{20}$$

where $\sigma = \|x\|_1/\sqrt{n}$ and ϵ is the machine epsilon. The optimality tolerance employed in (7) is equivalent to a feasibility tolerance for the original problem of:

$$c_i - a_i^T w \ge -10^{-6}, \qquad i = 1, \cdots, n. \tag{21}$$

Computational results for RMG are given in Table 1. For all problems, the number of nonlinear variables is $s = 10$. For each problem listed in Table 1, we give:

1. Blocks: the number of diagonal blocks in the matrix A.
2. m, n: the number of linear variables and constraints.
3. Seconds: the cpu time for solving the optimization problem.
4. Major iterations: the number of Newton steps in the reduced KKT equations.
5. $\|g\|$: the reduced gradient norm on termination.
6. $\|y - y^*\|$: the error norm for the nonlinear variables.

Table 1 gives results for a series of problems in which the number of nonlinear variables is fixed, and the number of linear variables and constraints is increased by a factor of 2, by doubling the number of diagonal blocks in A. Hence, the A matrix becomes increasingly large and sparse.

The reduced gradient and feasibility criteria were satisfied on all problems in Table 1 (most of the norms indicated in Table 1 are many orders of magnitude smaller than the termination criteria). Inspection of Table 1 shows that execution time increases with problem size. Some of the variance is also attributable to the line search process (19), which must solve (7) to do a function evaluation. If the accepted step length α_k varies substantially in magnitude from one major iteration to the next, a significant amount of time is used in function evaluation to locate the appropriate interval. To further analyze the results in Table 1, we calculated the average time per iteration for each problem. The average time per iteration (not shown) increases monotonically at a sublinear rate with the problem size. In contrast, total time for solving the problem

TABLE 1
Serial RMG *results on* CRAY-2.

Block size 10 x 20, 100 percent dense						
Blocks	m	n	Sec	Maj It	$\|g\|$	$\|y - y^*\|$
1	10	20	0.09	8	0.8E-12	0.4E-13
2	20	40	0.38	18	0.1E-11	0.4E-14
4	40	80	0.68	23	0.2E-11	0.4E-13
8	80	160	0.44	9	0.3E-11	0.5E-14
16	160	320	1.15	18	0.6E-11	0.5E-13
32	320	640	3.90	29	0.2E-10	0.1E-13
64	640	1280	6.82	29	0.3E-10	0.6E-13
Block size 50 x 100, 10 percent dense						
Blocks	m	n	Sec	Maj It	$\|g\|$	$\|y - y^*\|$
1	50	100	1.29	10	0.3E-11	0.6E-14
2	100	200	5.22	14	0.4E-11	0.7E-13
4	200	400	11.34	19	0.2E-10	0.2E-13
8	400	800	12.61	12	0.2E-10	0.4E-13
16	800	1600	26.81	20	0.3E-10	0.1E-13
32	1600	3200	31.95	21	0.5E-07	0.8E-05
64	3200	6400	91.35	37	0.8E-05	0.5E-06

is approximately bounded by a linear function of the problem size; i.e. execution time appears to approximately double as the problem size doubles.

6.2. Comparison with MINOS. For nonlinearly constrained problems, MINOS employs a projected augmented Lagrangian algorithm. This algorithm uses the reduced gradient method to solve a sequence of linearly constrained subproblems.

There are a number of important similarities and differences between MINOS and the RMG algorithm. Both RMG and MINOS partition linear and nonlinear variables, both compute and factorize a basis for the strictly linear part of the constraint coefficient matrix A, and both compute a form of the reduced Hessian matrix. RMG uses exact second derivatives to compute a Newton iteration in the reduced first-order (KKT) equations and uses the Newton step as a search direction for (19). MINOS uses quasi-Newton updates of the reduced Hessian and computes a projected Newton search direction for the reduced gradient subproblem. RMG requires that each iterate remain feasible for the original problem. MINOS does not insist on feasibility until a solution is found. Perhaps the most crucial difference is that given a structured set of linear coefficients (e.g, block-diagonal A), RMG evaluates the objective function as a separable LP, meaning that independent blocks of A are treated as independent LPs. MINOS does not treat special structures in the linear constraint coefficients.

MINOS 5.3 was implemented on the Minnesota Supercomputer Center CRAY-2 in single precision arithmetic, using CRAY-optimized BLAS and default MINOS parameters. All MINOS runs were made using one processor of the CRAY-2. Time measurements for MINOS reflect the time for subroutine M5SOLV. The termination criteria for MINOS include

$$\begin{aligned} g_i/\sigma &\le 10^{-6}, \qquad i = 1, \cdots, s, \\ (b_i - a_i^T x)/\sigma &\le 10^{-6}, \qquad i = 1, \cdots, m, \end{aligned} \tag{22}$$

where $\sigma = \|x\|_1/\sqrt{n}$. The feasibility tolerance is:

$$c_i - a_i^T w \ge -10^{-6}, \qquad i = 1, \cdots, n. \tag{23}$$

These criteria are comparable to those indicated above for RMG. The feasibility

tolerances are identical, while the reduced gradient tolerances differ by less than an order of magnitude from those employed with RMG.

Computational results for MINOS are given in Table 2, for the same set of problems as shown in Table 1. For all problems, the number of nonlinear variables is $s = 10$. For each problem listed in Table 2, we give:

1. Blocks: the number of diagonal blocks in the matrix A.
2. m, n: the number of linear variables and constraints.
3. Seconds: the execution time for subroutine M5SOLV.
4. Major iterations: the number of linearly constrained subproblems that were solved.
5. Minor iterations: the number of steps taken by the reduced gradient algorithm.
6. $\|rg\|$: the reduced gradient norm on termination.

TABLE 2
Serial MINOS *results on* CRAY-2.

Block size 10 x 20, 100 percent dense						
Blocks	m	n	Sec	Maj Itn	Min Itn	$\|rg\|$
1	10	20	0.36	13	126	1e-7
2	20	40	0.82	11	195	1e-8
4	40	80	1.72	10	265	2e-7
8	80	160	4.11	12	351	1e-7
16	160	320	12.63	17	562	1e-5
32	320	640	51.70	30	1080	2e-6
64	640	1280	252.67	58	2204	9e-6
Block size 50 x 100, 10 percent dense						
Blocks	m	n	Sec	Maj Itn	Min Itn	$\|rg\|$
1	50	100	2.72	12	318	2e-6
2	100	200	6.22	14	420	4e-6
4	200	400	20.20	21	722	6e-7
8	400	800	112.19	29	2068	6e-7
16	800	1600	575.57	33	4684	3e-6

The reduced gradient and feasibility criteria for MINOS were satisfied on all problems. For the larger problems, the minor iteration limit for each subproblem had to be increased beyond the default to obtain convergence. Execution time for MINOS increases approximately as a factor of 4 as the problem size doubles. The number of major iterations is a slowly growing function of the problem size, while the number of minor iterations is approximately linear in the problem size.

A comparison of MINOS and RMG shows that for the smallest problem (10×20) MINOS required 0.36 seconds compared to 0.09 seconds for RMG, while for the largest problem (800×1600) MINOS required 576 seconds compared to 27 seconds for RMG. Based on additional runs where A is unstructured (not shown), we attribute the performance differential primarily to the exploitation of structure by RMG; i.e., the evaluation of (7) as a separable linear program.

6.3. The parallel code. A motivation for the RMG algorithm is that the independent linear programs (7), corresponding to independent blocks of A, are suitable for asynchronous computation on individual processors of an MIMD architecture. With the supporting assumption that $s \ll n$, the work required to solve the LPs will tend to dominate the cost of the algorithm, even with a large number of processors. To test these ideas, a parallel RMG algorithm has been developed for MIMD architectures and implemented on a message-passing hypercube multiprocessor.

The MIMD algorithm assumes allocation of one or more blocks of constraints to each processor, where a block of constraints is defined in terms of the block-diagonal matrix A. Constraint and objective function evaluation (involving solution of (7)) are essentially parallel operations requiring a minimum of $O(mn)$ total operations per major iteration. The load-balancing characteristics of the algorithm depend on the similarity of block dimensions. A fixed (small) number of global communications are required for each major iteration. Each synchronization involves summing or comparing a scalar or s-vector among all processors. Global communication is implemented on a hypercube architecture with a spanning-tree algorithm (see, for example, [1]). Thus, if p is the number of processors, the cost of a global send or receive is proportional to $\log(p)\tau$, where τ is the sum of the latency time and transfer time for sending or receiving an s-vector between nearest neighbors. The major serial computation is the Newton trust-region iteration, which is currently a serial $O(s^3)$ computation performed redundantly by all processors.

The parallel implementation was tested experimentally on a first generation NCUBE hypercube maintained by the University of Minnesota High Performance Computing Laboratory. The NCUBE has 64 32-bit processors organized in a message-passing hypercube network, accessed through a SUN 3/60 front-end. The algorithm was implemented in double precision arithmetic. The CRAY and NCUBE codes are almost identical from a numerical point of view, except for minor differences in the criteria for sufficient improvement in the line search. Time measurements for the NCUBE are based on the cpu time for a timekeeper processor, which is the first to begin and last to end computation. The termination criteria are identical to the serial code.

Computational results for the MIMD RMG code are given in Table 3, for the same set of problems as shown in Table 1. For all problems, the number of nonlinear variables is $s = 10$. For each problem listed in Table 3, we give:

1. Proc: the number of processors.
2. m, n: the total number of linear variables and constraints.
3. Sec: the cpu time for the timekeeper processor.
4. $\|g\|$: the reduced gradient norm on termination.
5. $\|y - y^*\|$: the error norm for the nonlinear variables.

Table 3 illustrates an experiment similar to Tables 1 and 2, in which the number of nonlinear variables is fixed, and the number of linear variables and constraints is increased by a factor of 2, by doubling the number of diagonal blocks in A. For the NCUBE implementation, we have assigned one block of A per processor, so the number of processors doubles as the number of blocks.

All problems in Table 3 satisfied the reduced gradient and feasibility criteria on termination. The number of major iterations is in most cases identical to the serial CRAY results in Table 1. Differences in the number of iterations between Tables 1 and 3 arise due to minor differences in the line search implementations. The iteration paths of the two implementations tend to diverge toward the end of the process, when major iterations are relatively cheap and are of little significance in total time. The variability in execution time with respect to problem size (the number of blocks) was explained previously in connection with the CRAY results, and is more obvious in Table 3 because of the increase in processors with problem size. Execution time in Table 3 is a slowly growing (sublinear) function of problem size, a consequence of the corresponding increase in number of processors.

Finally, we give a brief comparative analysis of the serial and parallel RMG im-

TABLE 3
Parallel RMG *results on* NCUBE.

Block size 10 x 20, 100 percent dense						
Proc	m	n	Sec	it	$\|g\|$	$\|y - y^*\|$
1	10	20	7.09	8	2.2E-14	2.9E-16
2	20	40	21.72	18	1.4E-14	4.0E-15
4	40	80	22.84	20	4.9E-14	2.3E-16
8	80	160	13.03	9	8.0E-14	7.3E-16
16	160	320	19.12	18	2.7E-13	1.7E-15
32	320	640	39.49	28	3.2E-13	1.6E-15
64	640	1280	40.48	26	1.1E-12	1.8E-15
Block size 50 x 100, 10 percent dense						
Proc	m	n	Sec	it	$\|g\|$	$\|y - y^*\|$
1	50	100	156.49	10	1.3E-13	2.0E-15
2	100	200	203.07	14	2.7E-13	2.7E-15
4	200	400	269.17	19	5.0E-13	2.4E-16
8	400	800	228.57	12	9.3E-13	4.9E-16
16	800	1600	295.41	20	1.7E-12	3.1E-15
32	1600	3200	332.60	21	5.2E-08	8.2E-06
64	3200	6400	548.13	28	5.4E-12	3.2E-15

plementations. Comparing the one-block problems from Tables 1 and 3, we observe that the CRAY-2 code is approximately 100 times faster than the parallel code on a single NCUBE processor. For the largest problems in Tables 1 and 3, we observe that the CRAY-2 code is between 5 and 10 times faster than the NCUBE using all 64 processors.

7. Conclusions. For a class of model problems with known solution characteristics, it was found that RMG execution time was bounded by a linear function of the problem size. In contrast, execution time for MINOS increased approximately as a quadratic function of problem size. In both codes, the observed number of major iterations appears to grow slowly as a function of the problem size. The performance differential is attributed primarily to the exploitation of constraint structure by RMG.

RMG execution time for the same class of problems on a hypercube multiprocessor appears to be a slowly growing function of the problem size, attributable to a corresponding increase in the number of processors. A comparative analysis of serial and parallel execution times supports the conclusion that parallel efficiency is primarily a function of the number of nonlinear variables, and is not strongly affected by the number of linear variables.

REFERENCES

[1] S. L. JOHNSSON, *Communication efficient basic linear algebra computations on hypercube architectures*, Res. Report YALEU/DCS/RR-361, Computer Science Department, Yale University, New Haven, CT, 1985.

[2] L. S. LASDON, *Optimization Theory for Large Systems*, Macmillan, London, 1970.

[3] F. A. LOOTSMA AND K. M. RAGSDELL, *State-of-the-art in parallel nonlinear optimization*, Parallel Comput., 6 (1988), pp. 133–155.

[4] D. G. LUENBERGER, *Linear and nonlinear programming*, Addison–Wesley, Reading, MA, 1984.

[5] R. S. MAIER, *Parallel solution of large-scale optimization problems: A partition algorithm for linear programming*, Ph.D. thesis, Computer Science Department, University of Minnesota, Minneapolis, MN, 1990.

[6] B. MURTAGH AND M. SAUNDERS, *Large-scale linearly constrained optimization*, Math. Programming, 14 (1978), pp. 41–72.

[7] ———, MINOS 5.0 *user's guide*, Report SOL 83-20, Department of Operations Research, Stanford University, Palo Alto, CA, 1983.

[8] J. B. ROSEN, *Convex partition programming*, in Recent Advances in Mathematical Programming, R. L. Graves and P. Wolfe, eds., McGraw–Hill, New York, 1963, pp. 159–176.

[9] J. B. ROSEN AND R. S. MAIER, *Parallel solution of large-scale block-angular linear programs*, Ann. Oper. Res., 22 (1990), pp. 23–41.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 515–529, November 1991

006

PARALLEL GENETIC ALGORITHMS APPLIED TO THE TRAVELING SALESMAN PROBLEM*

PRASANNA JOG†, JUNG Y. SUH‡, AND DIRK VAN GUCHT‡

Abstract. *Genetic algorithms* are adaptive search algorithms that have been shown to be robust optimization algorithms for multimodal real-valued functions and a variety of combinatorial optimization problems. In contrast to more standard search algorithms, genetic algorithms base their progress on the performance of a population of candidate solutions, rather than on a single candidate solution.

The authors will concentrate on the application of genetic algorithms to the *traveling salesman problem*. For this problem, there exist several such algorithms, ranging from *pure* genetic algorithms to genetic algorithms that incorporate heuristic information. These algorithms will be reviewed and their performance contrasted.

A serious drawback of genetic algorithms is their inefficiency when implemented on a sequential machine. However, due to their inherent parallel properties, they can be successfully implemented on parallel machines, resulting in considerable speedup. *Parallel genetic algorithms* will be reviewed and their uses in the traveling salesman problem will be indicated.

Key words. genetic algorithms, traveling salesman problem, parallel algorithms

AMS(MOS) subject classifications. 05AXX, 05A05, 68TXX, 68T05, 90BXX, 90B40

1. Introduction.

Suppose we have an *object space* X and a function $f : X \to R^+$ (R^+ denotes the positive real numbers) and our task is to find a global optimum for that function. *Genetic algorithms* are a class of adaptive search algorithms invented by Holland [15] to solve (or approximately solve) such problems. Genetic algorithms differ from more standard search algorithms (e.g., gradient descent, controlled random search, hill-climbing, simulated annealing, etc.) in that the search is conducted using the information of a *population* of *structures* of the object space X instead of that of a single structure. The motivation for this approach is that by considering many structures as potential candidate solutions, the risk of getting trapped in a local optimum is greatly reduced. In Fig. 1 we show the layout of a typical genetic algorithm (GA). The initial population `P(0)` consists of structures of X, usually chosen at random. Alternatively, `P(0)` may contain heuristically chosen structures. In either case, the initial population should contain a wide variety of structures. Each structure x in `P(0)` is then evaluated by applying to it the function f. The genetic algorithm then enters a loop. Each iteration of that loop is called a *generation*. The new population `P(t+1)` is constructed in two steps: (1) the *selection* step and (2) the *recombination step*. In the selection step, a temporary population (say, $\mathtt{P}'(\mathtt{t}+1)$) is constructed by choosing structures in `P(t)` according to their relative performance. For example, if we are maximizing f, the structures with greater than average performance will be selected with higher probability than the structures with below average performance. This resembles the *survival of the fittest principle* of natural evolution. After the selection step, the temporary population $\mathtt{P}'(\mathtt{t}+1)$ is *recombined*. (The resulting population is the new population `P(t+1)`.) Typically, recombination is accomplished by applying several *recombination operators*, such as *crossover*, *mutation*, *inversion* [8], [15], or *local improvement* [38], to the structures in $\mathtt{P}'(\mathtt{t}+1)$. After the recombination

* Received by the editors August 17, 1990; accepted for publication (in revised form) March 14, 1991.

† Computer Science Department, De Paul University, Chicago, Illinois 60614.

‡ Computer Science Department, Indiana University, Bloomington, Indiana 47405-4101.

```
P(t) denotes the population at time t.
t ← 0;
initialize P(t);
evaluate P(t);
while (termination condition is not satisfied)
{
    t ← t+1;
    select P(t);
    recombine P(t);
    evaluate P(t);
}
```

FIG. 1. *Layout of a standard genetic algorithm.*

step is completed, the new population is reevaluated and a termination condition is checked for validity.

The difference between genetic algorithms and earlier-introduced evolutionary algorithms is the usage of crossover in the former. A *crossover* operation produces an offspring from two structures. It is typically defined such that a sufficient amount of information embedded in the parents appears in the offspring. Therefore, crossover can be viewed as a *history-preserving* operation which at the same time introduces a new structure to be tested in the competition. This is clearly different from a random mutation, which was the central operator in evolutionary algorithms.

Genetic algorithms have been applied to global function optimization, combinatorial optimization, machine learning, etc. [9]. In this paper, we will concentrate on the application of genetic algorithms to the *traveling salesman problem*. For this problem, there exist several different genetic algorithms, ranging from *pure* genetic algorithms to genetic algorithms which incorporate heuristic information. We will review these algorithms and contrast their performance (§2).

A serious drawback of genetic algorithms is their inefficiency when implemented on a sequential machine. However, due to their inherent parallel properties, they can be successfully implemented on parallel machines, resulting in considerable speedup. We will review *parallel genetic algorithms* in general, and subsequently indicate how they have been used in the traveling salesman problem (§3). In §4 we will show some additional results of applying parallel genetic algorithms to large TSPs (the largest problem is a 1000-cities problem). This section also contains data that indicates the relative importance of certain recombination operations.

2. Genetic algorithms for the traveling salesman problem. The traveling salesman problem (TSP) is easily stated: Given a complete graph with N nodes, find the shortest Hamiltonian tour through the graph (in this paper, we will assume Euclidean distances between nodes). For an excellent discussion of the TSP, we refer to [20].

For the TSP, the object space X consists of all Hamiltonian tours (tours, for short) and f, the function to be optimized, returns the length of tours. In recent years, a variety of GAs for the TSP have been proposed [3], [10], [9], [11], [14], [13], [17], [21], [25], [26], [27], [38], [37], [35]. These algorithms can be separated into two groups: (1) the *pure genetic algorithms*, i.e., algorithms that do *not* use domain-specific information about the TSP [10], [9], [13], [27], and (2) *heuristic genetic algorithms* (HGA), i.e., algorithms that do use domain-specific information about the TSP [3], [11], [14], [13], [17], [21], [25], [26], [38], [37], [35].

$A = 9\ 8\ 4 \| 5\ 6\ 7 \| 1\ 3\ 2\ 10$
$B = 8\ 7\ 1 \| 2\ 3\ 10 \| 9\ 5\ 4\ 6$

Two aligned tours

$A' = 9\ 8\ 4 \| 2\ 3\ 10 \| 1\ 6\ 5\ 7$
$B' = 8\ 10\ 1 \| 5\ 6\ 7 \| 9\ 2\ 4\ 3$

Result of the PMX crossover on the two above tours

FIG. 2. *Illustration of a* PMX *crossover.*

2.1. Pure genetic algorithms for the TSP. Pure genetic algorithms use recombination operators that apply to arbitrary permutations, hence they can be used in any problem domain involving objects represented as permutations. (From now on, we will use the term *tour* instead of permutation since we are only considering the TSP.)

- An example of a domain independent mutation operator is the *inversion* operator [15], [13]. This operator, for a given tour, simply reverses a randomly chosen subtour.
- We now consider various pure crossover operators. All these crossover operators share the property of preserving, in the offspring, subtours of the parents. The intuition is that in doing so, potentially good subtours from the parents are combined to obtain a possible better tour as an offspring.
  1. The *partially matched crossover* (PMX) [10], [9] starts by aligning two tours, subsequently two crossing sites are picked uniformly at random along the tours (see top part of Fig. 2). These two points define a matching section that is used to effect a cross through position-by-position exchange operations. Note that it might be necessary to change the original tours outside the matching section to guarantee that the new objects are again tours (the result of a PMX crossover is shown in the bottom part of Fig. 2). The *order crossover* operator [7], [6], [9], [27], [36] is a variant of the PMX crossover.
  2. The *cycle crossover* (CX) operator [7], [6], [9], [27], [36] performs recombination under the constraint that each city name come from one parent or the other.
  3. The Grefenstette crossover (Gref-X) [14], [13], [32] operator takes two tours and constructs an offspring as follows. Randomly choose a city as the current city for the offspring tour. Consider the four edges incident to the current city in the parents. Randomly select one of these four edges and include it in the offspring. (If none of the parental edges leads to an unvisited city, create an edge to a randomly chosen unvisited city.) Repeat this process until all cities have been visited. Consider the tours at the top of Fig. 3. A possible offspring of these tours is shown at the bottom of this figure.
  4. The Mulhenbeim crossover [25] uses a donor and receiver tour. From the donor a random substring is chosen. All nodes which are included in the string are deleted from the receiver. Then the substring of the donor is copied over and the remaining receiver nodes are inserted in the order they appear.

$A = 9\ 8\ 2\ 1\ 7\ 4\ 5\ 10\ 6\ 3$
$B = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10$

Two tours before applying the Gref-X

$B = 9\ 8\ 2\ 3\ 4\ 5\ 6\ 7\ 1\ 10$
A possible offspring after a Gref-X

FIG. 3. *Illustration of a* Gref-X *crossover.*

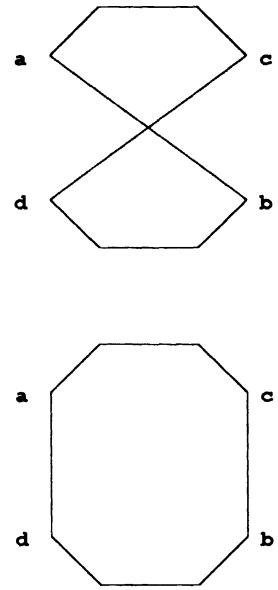


FIG. 4. *Illustration of a* 2-opt *operation.*

2.2. Heuristic genetic algorithms for the TSP. Heuristic genetic algorithms use recombination operators that incorporate heuristic information about the TSP. The essential heuristic in all these operators is the observation that solutions to the TSP contain short edges. Hence, when the opportunity arises to select between edges, the operators choose the shorter ones.

- Heuristic mutation is usually implemented in the form of so-called *local improvement operators*. Examples include the 2-opt and 3-opt operators of Lin and Kernighan [22] and the Or-opt operator of [20], [28].
  1. The 2-opt operator randomly selects two edges (a, b), (c, d) from a tour (see Fig. 4) and checks if $\mathrm{ED}(a, b) + \mathrm{ED}(c, d) > \mathrm{ED}(a, d) + \mathrm{ED}(c, b)$ (ED stands for Euclidean distance). If this is the case, the tour is replaced by removing the edges (a, b), (c, d) and replacing them with the edges (a, d), (c, b). (The 3-opt operator is similar to the 2-opt operator, except that it applies to three rather than two edges.)
  2. The Or-opt was introduced by Or [20], [28] and is a variant of the 3-opt operator. The advantage of the Or-opt operator is that it only considers a small percentage of the exchanges that would be considered by a regular 3-opt operator. To understand how the Or-opt procedure works, we refer to Fig. 5. For each connected string of s cities in the

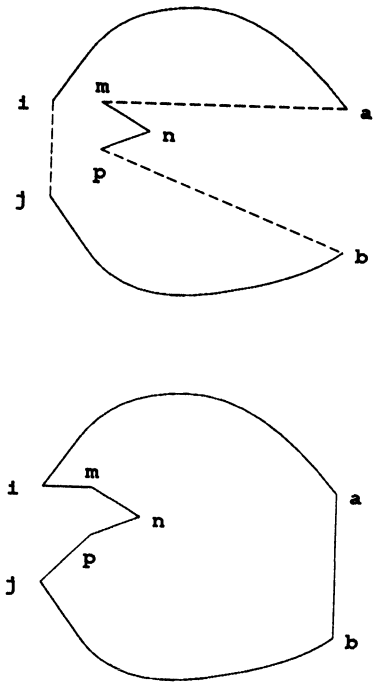


FIG. 5. *Illustration of an Or-*opt *local improvement operation.*

$$A = 9\ 8\ \|2\ 1\ 7\ 4\ 5\|10\ 6\ 3$$
$$B = 8\ 9\ \|2\ 7\ 5\ 1\ 4\|3\ 6\ 10$$

Two tours before applying the Brady crossover

$$C = 8\ 9\ 2\ 1\ 7\ 4\ 5\ 3\ 6\ 10$$

A possible offspring after a Brady crossover

FIG. 6. *Illustration of a Brady crossover operation.*

current tour (s can be 3, 2, or 1), we test to see if that string can be relocated between two other cities at reduced cost. If it can, we make the appropriate changes. For example, for $s = 3$ (see Fig. 5), we test to see if the string of the three adjacent cities m, n, p in the current tour is considered for insertion between a pair of connected cities i and j outside of the string. The insertion is performed if the total cost of the edges to be erased, $\{a, m\}, \{p, b\}$, and $\{i, j\}$, exceeds the cost of the new edges to be added, $\{i, m\}, \{p, j\}$, and $\{a, b\}$.

- Heuristic information has also been incorporated in various crossover operators for the TSP:
  1. The *Brady crossover* [3], [2] takes two tours and searches for subroutes where some common subset of cities are visited in a different order (see Fig. 6). The shorter subroute is then copied over to replace the longer one. In the figure, we assume that the sequence 2 1 7 4 5 is shorter than the sequence 2 7 5 1 4.
  2. The *Heuristic crossover* [14] operator constructs an offspring from two parent tours as follows: Pick a random city as the starting point for

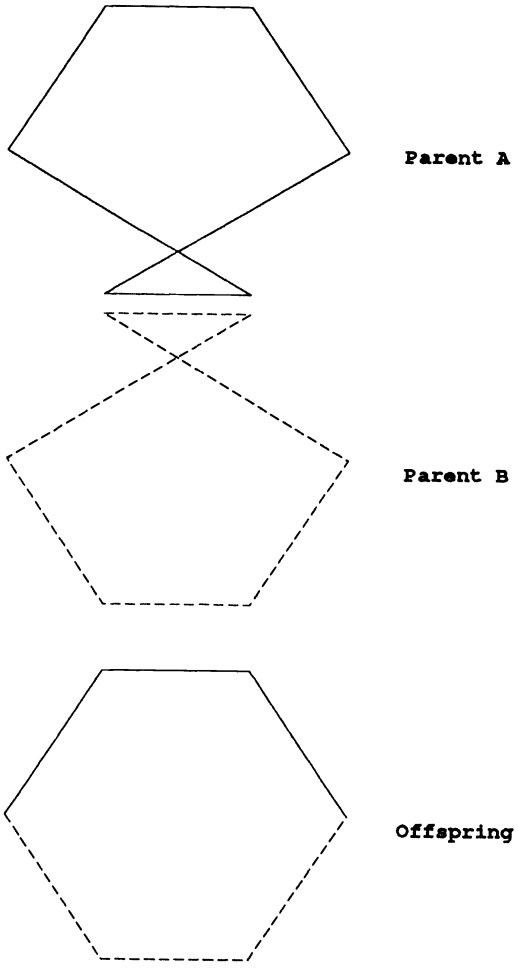


FIG. 7. *The Grefenstette crossover operator.*

the offspring's tour. Compare the two edges leaving the starting city in the parents and choose the shorter edge. Continue to extend the partial tour by choosing the shorter of the two edges in the parents which extend the tour. If the shorter parental edge would introduce a cycle into the partial tour, check if the other parental edge introduces a cycle. In case the second edge does not introduce a cycle, extend the tour with this edge, otherwise extend the tour by a random edge. Continue until a complete tour is generated (see Fig. 7). Variations of the Grefenstette crossover, including more heuristic information, were introduced by [17], [21], [38], and [35].

2.3. Performance results of genetic algorithms for the TSP. We now describe the performance results of the pure and heuristic genetic algorithms. We will indicate the size of the TSP problems attempted and the quality of the obtained solutions. Since most papers do not report the time required to run the algorithms, we can sometimes not report on this performance parameter. Instead, we will give the population size and the number of generations for the GAs to converge.

Reference	Algorithm	Problem	Population	Best	Average	Time
[9]	PMX	33-cities	2000	10.0	NA	300
[9]	Inversion	33-cities	2000	70.0	NA	300
[13]	Random	100-cities	100	480.0	NA	200
[13]	PMX	100-cities	100	210.0	NA	200

FIG. 8. *Results of pure genetic algorithms applied to the* TSP.

We first consider the performance of pure genetic algorithms. These results were obtained from [9], [13] and shown in Fig. 8. In this table, the column *algorithm* indicates the chosen recombination operators, *population* denotes the population size, *best* gives the percentage (above the known optimum for the TSP under consideration) of the best solution found by the GA, *average* gives the percentage above the optimum for a sample of solutions produced by the GA, *time* gives the number of generations. The 33-cities problem (100-cities problem) is a TSP with 33 (100) randomly distributed cities in a square.

As can be seen from this table, pure genetic algorithms perform poorly, especially when larger TSP problems are considered. We therefore do not consider such algorithms in the rest of the paper. The argument for this poor performance can be found in [13].

We next consider the performance of heuristic genetic algorithms. These results were obtained from [3], [11], [14], [13], [17], [25], [26], [38], [37], [35] and are shown in Fig. 9. We have organized the results chronologically since this reflects (1) how the problem sizes of attempted TSP problems increased, and (2) how the fine tuning of these algorithms led to better performance. The 50-cities (100-cities, 200-cities) problem is a TSP of 50 (100, 200) randomly distributed cities in a square. The optimum tour length for such problems can be estimated as described in [1]. The *Krolak* problem is the often-cited 100-cities problem described in [19]. This problem has been used in numerous studies as a benchmark ([20, Chap. 7]). The *lattice* problem is a TSP of 100 cities arranged in a ten-by-ten grid. This problem has been used as a benchmark in the performance of simulated annealing [4], [33]. The *Grotschel* problem is a TSP with 442 cities nonuniformly distributed in a square [33]. The *Padberg* problem is a TSP problem of 532 cities in the United States [29]. The *comments* column contains additional information about the particular GA used: Avg indicates from how many experiments the *Average*-column value is obtained. Low (moderate, high) 2-opt (Or-opt) indicates that the relative rate between 2-opt (Or-opt) operations trials and crossover operations is low (moderate, high). (For example, for a 100-cities problem, a typical low rate is in the range [10–20], a moderate rate in the range [100–200], and a high rate is in the 1000 and above range.) Parallel GA indicates that a parallel GA was used. Dynamic-X indicates that the characteristics of the crossover operator changed during the course of the algorithms.

As a general comment, it can be said that genetic algorithms are very good heuristic algorithms for the TSP. In several cases, the best solution was the optimum solution; in other cases, the performance was to within 1 percent of the optimum (this is especially the case for GAs which use a moderate to high ratio of 2-opt operations per crossover operator). Also notice how the robustness of the genetic algorithms is reflected in their average performance relative to their best performance. To even better appreciate the quality of these algorithms, we refer to Chapter 7 in [20], where a survey is given of the performance of many other heuristics for the TSP. Compared to this study, heuristic GAs compare well or outperform the best results obtained with

Ref.	Algorithm	Problem	Population	Best	Average	Time	Comments
[3]	2-opt Brady-X	64-cities	24(max)	NA	1.3	1.0s IBM 3081D	Avg 100
[14]	Heur-X	100-cities	100	16.0	27.0	400	
[35]	Inversion Heur-X	50-cities	10	5.5	NA	1000	Dynamic-X
[35]	Inversion Heur-X	100-cities	12	7.2	NA	3700	Dynamic-X
[35]	Inversion Heur-X	200-cities	15	3.4	NA	5800	Dynamic-X
[38]	2-opt Heur-X	krolak	100	1.8	3.7	500	Low 2-opt Avg 5
[38]	2-opt Heur-X	lattice	100	0.0	0.4	200	Low 2-opt Avg 5
[38]	2-opt Heur-X	200-cities	100	2.4	4.5	850	Low 2-opt Avg 5
[37]	2-opt Heur-X	krolak	100	1.2	2.3	400	Parallel GA Low 2-opt Avg 10
[37]	2-opt Heur-X	lattice	100	0.0	1.1	200	Parallel GA Low 2-opt Avg 10
[37]	2-opt Heur-X	200-cities	100	1.9	3.6	800	Parallel GA Low 2-opt Avg 10
[13]	Heur-X	100-cities	100	NA	< 7.4	200	Post 2-opt
[25]	2-opt Muhl-CX	krolak	100(max)	NA	1.9	NA	Parallel GA High 2-opt Avg 5
[26]	2-opt Muhl-CX	krolak	24	0.0	0.0	60	Parallel GA High 2-opt Avg 25
[26]	2-opt Muhl-CX	Grotschel 442-cities	24	1.0	NA	NA	Parallel GA High 2-opt
[17]	2-opt Or-opt Heur-X	krolak	100	0.0	1.4	590	Low 2-opt Low Or-opt
[17]	2-opt Or-opt Heur-X	lattice	100	0.0	0.4	420	Low 2-opt Low Or-opt
[11]	2-opt Muhl-CX	Grotschel 442-cities	64	0.3	0.4	NA	Moderate 2-opt Avg 10
[11]	2-opt Muhl-CX	Padberg 532-cities	64	0.2	0.4	1200	Moderate 2-opt Avg 10

FIG. 9. *Results of heuristic genetic algorithms applied to the* TSP.

the best heuristics reported there. When compared with simulated annealing, the GAs also performed well. For example, in [4] a 100 lattice problem is optimized using simulated annealing to a tour length about 3 percent above the desired optimum. GAs routinely solve this problem to optimality.

The table in Fig. 9 does not indicate all the features that were considered in GAs for the TSP. For example, some authors have studied the effects of the population size [3], [17], [25], [26], the initial diversity in the population [13], the quality of local improvement [17], the robustness of the GA (i.e., its ability to repeatedly find similar results) [11], [17], [16], and the usage of dynamic recombination operators [35], [38], [17].

3. Parallel genetic algorithms. Genetic algorithms, when implemented on a sequential machine, are notoriously slow. Luckily, genetic algorithms lend themselves naturally to speedup through parallelization. This fact led to the introduction of *parallel genetic algorithms* (PGA) [5], [11], [12], [16], [23], [25], [26], [24], [30], [31], [34], [38], [39], [40].

In addition to seeking speedup, this research has also pointed out some weaknesses in the design choices for genetic algorithms as originally proposed by Holland [15]:

- In accordance with nature, it is more natural to view a population as consisting of a set of independent structures, each with it own local behavior, i.e., each has the opportunity to initiate or undergo recombination operations, without the control of a global agent.
- Selection in standard GAs is a global process, i.e., selection of an individual depends on its performance relative to the average performance of the *entire* population. This is quite unlike nature and inefficient to parallelize [16]. Parallel GAs therefore introduce *local selection* without affecting the performance of the algorithm.
- PGAs are very reliable, especially when implemented in as distributed a fashion as possible, i.e., the failure of a processor usually does not affect the rest of the computation.
- PGAs allow for asynchronous behavior. This is not possible in standard GAs. This allows different structures to evolve at different speeds which may result in the global speedup of the algorithm, as well as the maintenance of diversity, a critical component for the success of a GA.

Parallel genetic algorithms can be categorized according to the level of distributedness of the population (coarse-grained versus fine-grained), and the manner in which the recombination and selection strategies are supported.

In a typical *coarse-grained* PGA, the population is divided into subpopulations. Each processor of the parallel machine gets allocated a subpopulation and independently runs a standard GA. In particular, the recombination and selection operations are performed within a subpopulation. To support global competition between the subpopulation, on an occasional basis, communication is established between the processors to facilitate selection between the subpopulations. As a side-effect of this global selection, individuals can migrate to other populations.

A typical *fine-grained* PGA is obtained by allocating a single individual[1] to each processor. Each processor is powerful enough to perform evaluations of individuals, and to perform recombination operations, such as mutation, local improvement, and crossover. Occasionally, communication is established between the processors to facilitate recombination and selection.

3.1. Coarse-grained parallel genetic algorithms. Pettey et al. [31] describe a coarse-grained PGA (implemented on an Intel iPSC, with an n-cube interconnection network, up to 16 processors) applied to De Jong's [8] test bed of five global function optimization problems. To support global selection, once every generation each processor sends its best individual to its neigboring processors and incorporates the best individuals, sent to it by its neigbors, into its local population. Best results were obtained with the maximum of 16 processors, each with a subpopulation

[1] Sometimes, instead of a single individual, a small number of individuals is allocated to each processor.

of size 50, and were comparable with results obtained with standard GAs. In [30], a theoretical analysis of this PGA is given.

Tanese [39] describes a coarse-grained PGA (implemented on a 64 NCUBE/six hypercube made by NCUBE Corporation) applied to a GA-hard global function optimization problem (see [39] for a definition of GA-hardness). To support global optimization, once every five generations, each processor sends a portion of its best individual to *one* of its neighbors. A fixed population of 400 individuals was distributed over 1 through 32 processors. The quality of the solution obtained with the PGA was similar to that of a standard GA. Furthermore, a nearly linear speedup was recorded. ([40] is a continuation of this study). An interesting addition is the introduction of a *partitioned* PGA, i.e., a PGA without migration. The author reports that this PGA is a better optimizer than a standard GA on the same problems, however, significant variations exists in the average performance of the subpopulations. To improve this, a reasonably small degree of migration was introduced both to yield the quality of the partitioned PGA and to obtain a comparable performance average amongst the subpopulation.

Cohoon et al. [5] describe a PGA which is a hybrid of the Pettey and Tanese algorithms. After a fixed amount of generations, each processor sends a set of its best individuals to all its neighbors. After receiving these individuals, each processor selects a new population on the basis of the old population and these new individuals. The chosen application problem is the quadratic assignment problem. In a simulation of the parallel algorithm on a sequential machine, the authors report a better quality of solutions for PGA than for a standard GA.

3.2. Fine-grained parallel genetic algorithms. Muhlenbein, Gorges-Schleuter and Kramer [25], [26] describe a fine-grained PGA and apply it to the traveling salesman problem. Their algorithm (see [25], [26]) is as follows:

1. Take a sequential heuristic algorithm for the TSP (such as 2-opt).
2. Give the problem to N processors with different starting configurations.
3. Each processor computes a local optimum according to the sequential heuristic algorithm.
4. Each process performs a local selection by assigning ranking weights to individuals in neighboring processors (better individuals receive higher weights and each processor has four neighbors). A mating partner is chosen probabilistically according to this weight distribution.
5. Perform crossover and mutation (see §2).
6. Reduce the problem by collapsing common subtours, solve the reduced problem and expand these solutions.
7. If not converged, go back to step 3.

The fine-grained nature of this algorithm is transparent in (a) steps 2, 3, and 6 where individual processors work independently towards a local minima, (b) step 4 where local selection is performed instead of global selection, as done in standard GAs, and (c) step 5 where crossover and mutation are done independently as local processes of the processors. This algorithm, and a refined version reported in [11] was successfully applied to various TSP problems (see Fig. 9). The work reported in [24] applies a similar fine-grained PGA to the quadratic assignment problem.

Suh and Van Gucht [37] describe a fine-grained PGA and apply it to the traveling salesman problem. Their approach considers a framework consisting of a pool of processors which execute identical or nearly identical tasks in parallel. Each processor has a local memory large enough to store a small number of structures, one

of which is called the *local structure.* The collection of all these local structures in the processors constitutes the population of the genetic algorithm. Each processor is capable of performing local tasks and communicating with the other processors. The local tasks and communication serve to (1) perform evaluation of individuals, (2) perform recombination operations, and (3) perform local selections (as described in [37], there exists a variety of reasonable local selection strategies). Their algorithm is very similar to the one of Muhlenbeim, Gorges-Scheutler, and Kramer, except that (1) instead of letting processors completely converge to local optima (as in step 3 of the algorithm in [25]) only a certain amount of local improvements is performed before processors can interact, and (2) communication can be established directly between each pair of processors, regardless of their geometric relationship within the connection network of the parallel machine. This algorithm was successfully applied to various TSP problems (see Fig. 9) and yields the same results as standard GAs. In §4 we will apply this algorithm to large TSP problems with different rates of local improvement operations.

Manderick and Spiessens [23] describe a fine-grained PGA very similar to the algorithms of [25], [26], [24], [11], and [37]. Each individual of the population is allocated to a single processor. The processors are interconnected in a grid. Each processor performs the evaluation of its individual. Mutation occurs locally within the processors. Local selection is performed by each processor and is implemented by the calculation of the fitness distribution in the neighborhood of that processor. Each processor selects a new individual on the basis of this distribution. Crossover is done between neighboring processors. A simulation of this algorithm was applied to global function optimization problems [8], [39] and a comparison was made with a standard GA. The quality of various performance measures showed only minor differences between the PGA and the standard GA.

Sannier and Goodman [34] describe a fine-grained PGA allocating a single or small number of individuals to each processor. The processors are interconnected in a grid. Communication can be established between each pair of processors, but the likehood of this depends inversely on the distance in the grid between the processors. This communication is used to perform inter-processor selection and crossover. This algorithm was applied to a survival game in which individuals compete for food.

4. Parallel algorithms applied to large TSPs. A closer look at Fig. 9 reveals that an important calibration factor in determining the quality of solutions obtained for various TSPs is the amount of local improvement operations performed on a tour during each generation.

The intuition is as follows: if too little local improvement is performed, the selection pressure of the GA will make it prematurely converge into a subquality local optimum. If too much local improvement is performed, however, there will be few chances for tours to crossover and selection will have few opportunities to weed out poor tours. This will unnecessarily increase the convergence time.

It is therefore reasonable to state that a balance has to be stricken between the desired quality of the solution and the time in which it is obtained. For example, the work of [38], [37], [17] emphasizes speed, whereas the work of [25], [26] emphasizes quality of solution. The best balance can be found in the work of [11], in which high quality solutions are obtained in the presence of plentiful use of the recombination operations and local selection.

The natural question arises as to what degree one can decrease the amount of local improvement operations per generation and still obtain high quality solutions.

In this section, we address this question by applying the parallel genetic algorithm of [37] to a variety TSPs of growing size.[2] Besides the krolak and padberg problems (see §2), we have considered:

- The hamiltonian *tour* 318-cities problem of nonuniformly distributed cities of [22]. The optimum solution for this problem is reported in [29].
- The lattice-400 problem, consisting of 400 cities arranged in a 20-by-20 grid. This problem has been studied in the context of simulated annealing [18].
- The 1000-cities problem of 1000 randomly distributed cities in a square. Problems of this size have been studied in the context of simulated annealing. To our knowledge, this is the largest problem yet attempted with a genetic algorithm.

In the top part of Fig. 10, we show the results of experiments run with a parallel version, in the style of [37], of the genetic algorithm with 2-opt and Or-opt of [17]. The *low* column contains the results of this algorithm with a low amount of local improvement operation per generation. The *moderate* column contains the results of this algorithm with a moderate amount of local improvement operation per generation. The x column indicates the amount of local improvements in the following sense. If $x = 0.1$ and we are working on a TSP of size s, then $0.1 * s$ local improvement operations (both 2-opt and Or-opt) are performed on each tour per generation. Clearly, x is a tuning parameter and was set in the range of [0.1,0.2] ([1.0,3.0]) for low (moderate) amounts of local improvements. (These ranges were determined experimentally.) The *best* (*average*) column contains the percentage away from the optimum of the best (average) solution found by the PGA. The *time* columns give the time in seconds of the PGA run with the maximum number of available processors (typically 70).[3] This figure clearly indicates the thesis that more local improvement per generation improves the quality of the solutions, but clearly at the expense of extra time. The termination condition we used was running a specified number of generations. A typical 400- (1000-) cities problem was run for 1000 (2500) generations. However, in all cases convergence occured earlier.

It could be asked what the limit behavior is of a PGA with an "unlimited" amount of local improvements between generations. Such algorithms are PGAs without selection and crossover. We call such algorithms *independent strategies*, because they consist of running, in parallel, independent sequential local searches. In the bottom part of Fig. 10, we show the results of the *independent strategy* on a selection of TSPs. Whereas on the smaller TSPs (i.e., size 100), the results are not significantly different, on the larger problems (above 500 cities), it is clear that crossover and selection do play a critical role in the performance of GAs for the TSP. It is also interesting to observe that on larger TSPs, PGAs with a low amount of local improvement yield better solutions than the independent strategies and this with better efficiency.

Finally, a word about speedup. Our PGA is coded so that it can be simulated on a varying number of processors. This allows us to measure the speedup of the algorithm as a function of the number of processors. Clearly, the more available processors are, the closer our implementation comes to a true fine-grained distributed algorithm. The speedup curves for various TSPs are nearly linear up to 90 processors.

[2] The algorithms were run on the BBN Butterfly at the Iowa State University. We were able to use up to 90 processors.

[3] In this respect, we would also like to point out that the majority of the time is spent in calculating double float square roots since for the large TSPs we were not able to distribute the distance matrix to each processor. If this could be done, all the algorithms would run a factor of 4 faster (this is a conservative estimate).

Problem	Low					Moderate			
	x	Best	Avg	Time(in sec)		x	Best	Avg	Time(in sec)
lattice(100)	0.1	0.8	2.3	17		1.0	0.8	1.2	42
krolak(100)	0.2	0.5	2.1	17		3.0	0.0	0.5	70
318-cities	0.2	2.9	4.6	275		1.6	2.0	2.9	730
400-lattice	0.15	1.9	2.2	310		1.2	0.4	1.0	1060
padberg(532)	0.2	2.9	3.8	470		0.5	1.6	2.4	1070
1000-cities	0.2	2.9	3.1	2700		1.2	1.0	1.6	7000

Parallel genetic algorithms with low and moderate local improvement for various TSPs

Problem	Best	Avg	Time
lattice(100)	1.7	1.9	33
krolak(100)	0.0	0.2	73
padberg(532)	7.4	8.1	490
1000-cities	4.0	4.2	5500

The independent strategy for various TSPs

FIG. 10. *Performance of parallel genetic algorithms.*

5. Conclusion. In this paper, we have given a review of parallel genetic algorithms applied to the traveling salesman problem. Our main conclusion is that such algorithms make excellent approximation algorithms for this problem even when compared with the best sequential heuristic algorithms. This is especially the case on large TSPs. Since the PGAs lend themselves naturally to parallelization, such algorithms can also be efficiently run on parallel machines.

This paper, however, leaves open some research issues:

- As indicated, the crossover plays a role in the quality of tours obtained with PGAs. It would be interesting to more thoroughly study its effect. As mentioned before, we expect the role of crossover to gain importance with growing-size TSPs.
- Even PGAs which use low amounts of local improvement can yield good results. An interesting question would be to determine how small we can make this amount (thereby improving the efficiency of the algorithm) and still get reasonable tours. A related issue was suggested by one of referees, i.e., varying the amount of selection pressure might allow the runs with differing amounts of local improvements to be completed in the same total computation time, thus answering the question of which method is best given a particular amount of time available.
- Although we have purposely selected TSPs with different structural properties (for example, compare the random cities problems and the lattice problems) we did not thoroughly determine if these properties play a critical role in the quality of the PGAs. We conjecture that indeed such structural properties can be important in the outcome of the algorithms.

Acknowledgment. We thank the referees for their constructive comments on an earlier version of this paper.

REFERENCES

[1] J. BEARDWOOD, J. HALTON, AND J. HAMMERSLEY, *The shortest path through many points*, Proc. Cambridge Philos. Soc., 55 (1959), pp. 299–327.

[2] D. BOUNDS, *New optimization methods from physics and biology*, Nature, 329 (1987), pp. 215–219.

[3] R. BRADY, *Optimization strategies gleaned from biological evolution*, Nature, 317 (1985), pp. 804–807.

[4] V. CERNY, *Thermodynamical approach to the traveling salesman problem*, J. Optim. Theory Appl., 45 (1985), pp. 41–52.

[5] J. COHOON, S. HEDGE, W. MARTIN, AND D. RICHARDS, *Punctuated equilibria: A parallel genetic algorithm*, in Proc. Second Internat. Conference on Genetic Algorithms and their Applications, Cambridge, MA, J.J. Grefenstette, ed., July 1987, pp. 148–154.

[6] L. DAVIS, *Applying adaptive algorithms to epistatic domains*, in Proc. Ninth IJCAI, Aug. 1985, pp. 162–164.

[7] ———, *Job shop scheduling with genetic algorithms*, in Proc. Internat. Conference on Genetic Algorithms, 1985, pp. 136–140.

[8] K. DEJONG, *Adaptive system design: A genetic approach*, IEEE Trans. Systems Man Cybernet., 10 (1980), pp. 556–574.

[9] D. GOLDBERG, *Genetic Algorithms in Serahc, Optimization and Machine Learning*, Addison-Wesley, New York, 1988.

[10] D. E. GOLDBERG AND R. LINGLE, *Alleles, loci, and the traveling salesman problem*, in Proc. Internat. Conference on Genetic Algorithms and Their Applications, J. J. Grefenstette, ed., July 1985, pp. 154–159.

[11] M. GORGES-SCHLEUTER, *Asparagos an asynchronous parallel genetic optimization strategy*, in Proc. Third Internat. Conference on Genetic Algorithms, J. Schaffer, ed., Morgan Kaufmann, Los Altos, CA, July 1989, pp. 422–427.

[12] J. J. GREFENSTETTE, *Parallel adaptive algorithm for function optimization*, Tech. Report CS-81-9, Computer Science Department, Vanderbilt University, Nashville, TN, Nov. 1981.

[13] ———, *Incorporating problem specific knowledge into genetic algorithms*, in Genetic Algorithms and Simulated Annealing, L. Davis, ed., Morgan Kaufmann, Los Altos, California, 1987.

[14] J. J. GREFENSTETTE, R. GOPAL, B. J. ROSMAITA, AND D. VAN GUCHT, *Genetic algorithms for the traveling salesman problem*, in Proc. Internat. Conference on Genetic Algorithms, J. J. Grefenstette, ed., July 1985, pp. 160–168.

[15] J. H. HOLLAND, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[16] P. JOG, *Parallelization of probabilistic sequential search algorithms*, Ph.D. thesis, Computer Science Department, Indiana University, Bloomington, IN, 1989.

[17] P. JOG, J. SUH, AND D. VAN GUCHT, *The effect of local improvement, selection and crossover on a genetic algorithm for the traveling salesman problem*, in Proc. Internat. Conference on Genetic Algorithms, J. J. Grefenstette, ed., Morgan Kauffman, Los Altos, CA, 1989.

[18] S. KIRKPATRICK, *Optimization by simulated annealing: Quantitative studies*, J. Statist. Phys., 34 (1983), pp. 975–986.

[19] P. D. KROLAK, W. FELTS, AND G. MARBLE, *A man-machine approach toward solving the traveling salesman problem*, Comm. ACM, 14 (1971), pp. 327–334.

[20] E. LAWLER, J. LENSTRA, A. RINNOOY KAN, AND D. SHMOYS, *The Traveling Salesman Problem*, Wiley–Interscience, New York, 1985.

[21] G. LIEPINS AND HILLIARD, *Greedy genetics*, in Proc. Second International Conference on Genetic Algorithms and Their Applications, J. Grefenstette, ed., July 1987, pp. 90–99.

[22] S. LIN AND B. KERNIGHAN, *An efficient heuristic algorithm for the traveling salesman problem*, Oper. Res., 21 (1973), pp. 498–516.

[23] B. MANDERICK AND P. SPIESSENS, *Fine-grained parallel genetic algorithms*, in Proc. Third Internat. Conference on Genetic Algorithms, J. Schaffer, ed., Morgan Kaufmann, July 1989, Los Altos, CA, pp. 428–433.

[24] H. MUHLENBEIN, *Parallel genetic algorithms, population genetics and combinatorial optimization*, in Proc. Third Internat. Conference on Genetic Algorithms, J. Schaffer, ed., Morgan Kaufmann, July 1989, pp. 416–421.

[25] H. MUHLENBEIN, M. GORGES-SCHEULTER, AND O. KRAMER, *Evolution algorithms in combinatorial optimization*, Parallel Comput., 4 (1987), pp. 269–279.

[26] ———, *Evolution algorithms in combinatorial optimization*, Parallel Comput., 7 (1988), pp. 70–85.

[27] I. OLIVER, D. SMITH, AND J. HOLLAND, *A study of permutation crossover operators on the traveling salesman problem*, in Proc. Second Internat. Conference on Genetic Algorithms and Their Applications, J.J. Grefenstette, ed., July 1987, pp. 224–230.

[28] I. OR, *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*, Ph.D. thesis, Northwestern University, Evanston, IL, 1976.

[29] W. PADBERG AND G. RINALDI, *Optimization of a 532-city symmetric tsp*, Oper. Res. Lett., 6 (1987), pp. 1–7.

[30] C. PETTEY AND M. LEUZE, *A theoretical investigation of a parallel genetic algorithm*, in Proc. Third International Conference on Genetic Algorithms, J. Schaffer, ed., Morgan Kaufmann, July 1989, Los Altos, CA, pp. 398–405.

[31] C. PETTEY, M. LEUZE, AND J. GREFENSTETTE, *A parallel genetic algorithm*, in Proc. Second International Conference on Genetic Algorithms and their Applications, J.J. Grefenstette, ed., July 1987, pp. 155–161.

[32] B. ROSMAITA, *Exodus, an extension of the genetic algorithm to deal with permutations*, Master's thesis, Computer Science Department, Vanderbilt University, Nashville, TN, 1985.

[33] Y. ROSSIER, M. TROYON, AND T. LIEBLING, *Probabilistic exchange algorithms and the euclidean traveling salesman problem*, OR Spektrum, 8 (1986), pp. 151–164.

[34] A. SANNIER AND E. GOODMAN, *Genetic learning procedures in distributed environments*, in Proc. Second Internat. Conference on Genetic Algorithms and Their Applications, J.J. Grefenstette, ed., July 1987, pp. 162–169.

[35] D. SIRAG AND P. WEISSER, *Toward a unified thermodynamic genetic operator*, in Proc. Second Internat. Conference on Genetic Algorithms and Their Applications, J.J. Grefenstette, ed., July 1987, pp. 116–122.

[36] D. SMITH, *Bin packing with adaptive search*, in Proc. Internat. Conference on Genetic Algorithms and Their Applications, J.J. Grefenstette, ed., July 1985, pp. 202–206.

[37] J. SUH AND D. VAN GUCHT, *Distributed genetic algorithms*, Tech. Report 225, Computer Science Department, Indiana University, Bloomington, IN, July 1987.

[38] ———, *Incorporating heuristic information into genetic search*, in Proc. Second Internat. Conference on Genetic Algorithms and their Applications, J.J. Grefenstette, ed., July 1987, pp. 100–107.

[39] R. TANESE, *Parallel genetic algorithms for a hypercube*, in Proc. Second Internat. Conference on Genetic Algorithms and their Applications, J.J. Grefenstette, ed., July 1987, pp. 177–183.

[40] ———, *Distributed genetic algorithms*, in Proc. Third Internat. Conference on Genetic Algorithms, J. Schaffer, ed., July 1989, Morgan Kaufmann, Los Altos, CA, pp. 434–440.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 530–547, November 1991

007

A GENERAL-PURPOSE PARALLEL ALGORITHM FOR UNCONSTRAINED OPTIMIZATION*

STEPHEN G. NASH† AND ARIELA SOFER†

Abstract. This paper describes a general-purpose algorithm for unconstrained optimization that is suitable for a parallel computer. It is designed to be as easy to use as traditional algorithms for this problem, requiring only that a (scalar) subroutine be provided to evaluate the objective function and its gradient vector of first derivatives. The algorithm used is a block truncated-Newton method. Truncated-Newton methods are a class of methods that compromise on Newton's method so that large problems can be solved. Enhancements to the basic method suitable for a parallel computer are described. These include a revised data storage scheme, new preconditioning and initialization strategies to accelerate the method, a parallel line search, revised stopping rules for the inner algorithm, and a new "nonlinearity" test to determine the adequacy of the quadratic model. Numerical results are presented to illustrate the performance of the method, and comparisons are made with other scalar and parallel algorithms.

Key words. nonlinear optimization, truncated Newton method, parallel computing, conjugate-gradient method

AMS(MOS) subject classifications. 90C06, 65Y05

1. Introduction.

We discuss here the solution of the problem

$$\text{minimize } f(x) \tag{1.1}$$

on a parallel computer. The function $f(x)$ is a real-valued function of n variables x, and we are mainly interested in the case where n is large. We assume that $f(x)$ has at least two continuous derivatives. In addition, we assume that each processor of the parallel computer is capable of evaluating $f(x)$ and its gradient of first derivatives $g(x) \equiv \nabla f(x)$. Our numerical results were obtained using a hypercube computer with local memory, but the algorithm we describe has also been implemented on shared-memory machines.

Our goal was to produce an effective *general-purpose* algorithm for a parallel computer that is as easy to use as a traditional algorithm on a sequential computer (as easy to use as, say, a quasi-Newton method). We require that a subroutine be provided to evaluate $f(x)$ and its gradient $g(x)$ at an arbitrary point, but not any other function information. In particular, it is not necessary for the user to indicate how to decompose the function evaluation so that the work can be distributed over the parallel computer. The algorithms we will present will work for an arbitrary number of processors (even as low as 1), but are likely to be most effective when the number of processors is relatively small, say, less than 100, and when the number of variables is considerably greater than the number of processors.

For exceedingly difficult optimization problems, it may still be necessary to analyze the function-evaluation algorithm so that it too can exploit the parallel architecture. However, in less desperate circumstances, particularly in exploratory work where

*Received by the editors August 3, 1990; accepted for publication (in revised form) March 5, 1991. This work was supported by Center for Innovative Technology grant INF-90-004.

†Operations Research and Applied Statistics Department, George Mason University, Fairfax, Virginia 22030. The work of the second author was supported by National Science Foundation grant ECS–8709795.

various optimization models are being conjectured, the effort required to hand-code the optimization model for the parallel computer will not be cost-effective, and use of the parallel computer with its potentially greater power will be discouraged, especially if the user has not had prior experience with parallel computing. The hope is that algorithms of the type described in this paper will allow *routine* use of a parallel computer for optimization, even by "naive" users.

We would also hope that such an algorithm would be useful if the evaluation of the objective function does not naturally decompose, or does not decompose into subtasks of comparable size, so that there is not a natural mapping of the function evaluation to the parallel computer.

To solve (1.1) we will use a block truncated-Newton method. Truncated-Newton methods are a class of methods that compromise on Newton's method so that large problems can be solved. The compromises reduce the amount of storage needed (matrix storage for Newton's method versus vector storage here), function information (second derivatives versus first derivatives), and work per iteration ($O(n^3)$ versus $O(n)$–$O(n^2)$). A more detailed description is given in the next section.

Other general-purpose approaches to parallel optimization can be based on finite differencing in parallel [1] and on variants of quasi-Newton methods [2]. It might also be possible to adapt techniques of automatic differentiation [10] to use on a parallel computer. For a more comprehensive survey of parallel optimization, see the review papers [27] and [29].

The software that evaluates the function $f(x)$ and its gradient is almost always prepared independently of the software to solve the optimization problem (1.1). For difficult problems, the cost of the function evaluation can dominate the other costs of solving the problem. Hence, on a parallel computer, it is not sufficient merely to make parallel the internal steps of the optimization algorithm. To be effective on general problems, the optimization algorithm must be able to make use of *simultaneous* evaluations of the objective function. The block structure within our algorithm was introduced to obtain this property.

The basic algorithm described in this paper, a block truncated-Newton method, has been discussed in a previous paper [18], and was shown to be effective at exploiting the resources of a parallel computer. The purpose here is to describe enhancements to the algorithm that attempt to make it a competitive method for solving (1.1). These enhancements include: (a) reorganizing the algorithm to reduce communication costs, (b) designing automatic preconditioning and initialization schemes, (c) choosing appropriate rules for assessing the quality of a search direction, (d) incorporating a parallel line search, (e) a "nonlinearity" test to assess the accuracy of the Taylor-series approximation to the objective function, and (f) automating rescaling strategies that reset the preconditioner when it appears to be "out of date." With the exception of (c) and (d), these enhancements are new to this paper.

The resulting algorithm was tested on about sixty test functions having from 100–1000 variables. It was able to solve all the test problems, in most cases with significant reductions in solution times and numbers of gradient evaluations (measured per processor) over other algorithms. The most dramatic speedups were obtained on problems that were difficult for the scalar methods. However, improved performance was not obtained on all problems.

Here is an outline of the paper: In § 2, the basic algorithm is described. In § 3, the enhancements to the algorithm are discussed. In § 4, we report numerical results.

A software package based on the results in this paper is described in [20]. It includes the software used here for a hypercube (distributed-memory) parallel computer,

as well as a version for Sequent shared-memory parallel computers. This software is available from the authors.

In this paper, we have assumed that only gradient values are available (not second-derivative information), and that there is no special structure in the objective function. These assumptions were made so that the algorithm would be general purpose, and so that direct comparisons could be made with other algorithms for unconstrained optimization. If further information about the objective function is available, then the algorithm can be modified to take advantage of it. Such information might include: (a) exact second-derivative information (so that explicit matrix-vector products could be computed), especially if the Hessian were sparse or the function partially separable, (b) a function evaluation that could be performed in parallel. The associated software package [20] can take advantage of such information, although this option has not been used here.

2. Block truncated-Newton methods. We describe here the basic method that we will use to solve (1.1). Since this has been described in greater detail in [18], only an outline of the method will be given.

The algorithm is a descent method based on a line search. Given an initial guess x_0, at the kth iteration a search direction p is computed that satisfies $p^T g(x_k) < 0$; i.e., p is a "descent" direction for $f(x)$. A new point x_{k+1} is obtained via a line search, $x_{k+1} = x_k + \alpha p$, where $\alpha > 0$ is chosen so that $f(x_{k+1}) < f(x_k)$. Mild conditions on p and α guarantee convergence of the algorithm to a stationary point of $f(x)$ [25]. Various line search algorithms are described in [7]. A specific line search algorithm will be discussed in the next section.

It remains to describe the selection of the search direction p. In Newton's method p is computed as the solution to the system of linear equations

$$Gp = -g, \tag{2.1}$$

where $g = g(x_k)$ and $G \equiv \nabla^2 f(x_k)$ is the Hessian matrix of second derivatives at x_k. If G is not positive definite, a modified system of equations should be solved [8]. Newton's method converges quickly (an asymptotic quadratic rate of convergence) but is expensive (requiring second derivatives of $f(x)$, matrix storage, and the solution of a linear system at each major iteration, involving $O(n^3)$ arithmetic operations).

Truncated-Newton methods instead obtain an approximate Newton direction as an approximate solution to this system of equations. Some iterative method is applied to (2.1), but then is stopped ("truncated") before the exact solution has been obtained. The resulting method is doubly iterative: an "outer" iteration corresponding to the descent method above, and an "inner" iteration to compute the search direction. Traditionally, the linear conjugate-gradient method is used for the inner iteration.

Why bother? The resulting method can have rapid convergence rates, even quadratic, if desired [5]; the storage costs are low (just vector storage); only first derivatives are needed [23]; the method can be a robust, effective, and competitive general-purpose algorithm [15]; the search directions are well scaled, making the line search easy [14]; automatic preconditioning strategies can be derived [14]; and the algorithm vectorizes well [30]. Hence all the costs of Newton's method are reduced, and the resulting method is still good.

Truncated-Newton methods can be adapted to a parallel computer through an appropriate choice of algorithm for the inner iteration. We have chosen to use the block conjugate-gradient method, implemented in terms of the equivalent block-Lanczos method [22]. The algorithm corresponds to minimizing $Q(p) = \frac{1}{2} p^T G p + p^T g$ as a

function of p over a sequence of subspaces of increasing dimension (this is equivalent to solving (2.1) if G is positive definite). The description of the algorithm is adapted from [18].

Let G be an $n \times n$ symmetric matrix and let m be the block size. The block-Lanczos method computes a sequence of $n \times m$ orthogonal matrices $\{ V_i \}$ via:

Pick V_1 so that $V_1^T V_1 = I_{m \times m}$ and let $V_0 = 0_{n \times m}$, $\beta_1 = 0_{m \times m}$.
For $i = 1, 2, \cdots$
Compute $\alpha_i = V_i^T G V_i$, and let

$$V_{i+1}\beta_{i+1} = GV_i - V_i\alpha_i - V_{i-1}\beta_i^T,$$

choosing the $m \times m$ matrix β_{i+1} to make $V_{i+1}^T V_{i+1} = I_{m \times m}$.

Define $V_{(i)} = [V_1 | V_2 | \cdots | V_i]$. If we ignore rounding errors, then $V_{(i)}^T V_{(i)} = I$, and $V_{(i)}^T G V_{(i)} \equiv T_{(i)}$, where $T_{(i)}$ is a block tridiagonal matrix with blocks of size $m \times m$.

The block-Lanczos method is equivalent to the block conjugate-gradient method [22] for solving (2.1) if the first column of V_1 is chosen as $g / \|g\|_2$, where g is the right-hand side in (2.1). Now let $y_{(i)}$ be the solution of

$$T_{(i)} y_{(i)} = -V_{(i)}^T g = -\|g\|_2 e_1, \qquad e_1 = (1, 0, \cdots, 0)^T.$$

Then $p_{(i)}$, the ith approximation to the solution of (2.1), is given by $p_{(i)} = V_{(i)} y_{(i)}$. Also, $p_{(i)}$ minimizes the quadratic model $Q(p)$ over the subspace spanned by the columns of $V_{(i)}$. With the addition of a change of variables similar to that in [26], we obtain a practical iterative algorithm [18].

The matrix V_{i+1} may only have rank $m_1 < m$. As a result, β_{i+1} will be an $m \times m_1$ matrix, V_{i+1} will be an $n \times m_1$ matrix, etc. Except for these adjustments in matrix sizes, the algorithm is otherwise unchanged. A similar situation can occur at the final iteration in cases when m does not divide n, if the algorithm converges early, or if there is loss of orthogonality or linear independence due to rounding errors.

If the matrix G is not positive definite, the computation of the search direction must be modified. (The solution of (2.1) might not be a descent direction, and numerical instabilities can arise.) Special factorizations have been suggested [7], [13], [24], [26], but a simpler approach is used here. Note that the exact solution of (2.1) is not required, instead only a search direction is. In our software we terminate the iteration if indefiniteness is detected. The search direction is obtained from the partial factorization of $T_{(i)}$, up to but not including the row where indefiniteness is detected.

The implementation described here assumes that m (the block size, the number of processors) is small, and stores complete copies of the $m \times m$ matrices α, β, etc., on each processor. All computations involving these matrices are carried out simultaneously on all processors to reduce communication costs.

Preconditioning strategies can be used to enhance the algorithm. If $M \approx G(x_k)$ and if linear equations of the form $My = z$ are "easy" to solve, then M can be used to accelerate the convergence of the (block) linear conjugate-gradient method [3], [22]. (Choices of M are described in §3.2.) For the block-Lanczos algorithm, the "preconditioned" iterative formula becomes

$$V_{i+1}\beta_{i+1} = GR_i - V_i\alpha_i - V_{i-1}\beta_i^T,$$

where $R_i = M^{-1} V_i$, $\alpha_i = R_i^T G R_i$, and β_{i+1} is chosen so that $V_{i+1}^T R_{i+1} = I$. No other changes in the algorithm are required.

In this block algorithm the major computation at each inner iteration is a set of matrix-vector products involving the Hessian: Gv_i, for $i = 1, \cdots, m$, where m is the block size. Each of these can be performed simultaneously, one per processor, and can be computed via the finite-difference approximation

$$Gv \approx \frac{g(x_k + hv) - g(x_k)}{h}. \tag{2.2}$$

Note that $g = g(x_k)$ is already available. Hence each matrix-vector product can be approximated at the cost of one gradient evaluation. More importantly, *the algorithm is exploiting simultaneous gradient evaluations, one per processor.* This observation is the basis of the parallel algorithm.

The remaining operations of the inner algorithm also can exploit the resources of the parallel computer [16]. The lower-level operations of the algorithm correspond to standard vector operations (primarily basic linear algebra subroutines (BLAS)), and hence the algorithm can exploit the resources of parallel computers having special-purpose vector processors (such as vector hypercubes and Alliant computers).

This basic algorithm can be effective on a parallel computer. On some problems in [18] speedups of as much as 50 were obtained, even though only 16 processors were available. Note that the block algorithm is not a parallel version of the scalar method. It computes different search directions, and hence should be considered a different algorithm (so "speedup" does not compare two versions of the same algorithm, but instead the relative performance of two algorithms). Dramatic speedups such as this were not obtained on all problems. The goal of the work in this paper is to enhance the basic algorithm so that effective performance can be obtained on a much wider class of problems. The enhancements we have investigated are described in the next section.

3. Enhancements to the basic algorithm. There are many ways to improve the performance of the basic algorithm, both at the level of the specific (the actual computer implementation) and the general (the choice of line search). These are discussed below.

3.1. Organization of the algorithm. On a distributed-memory computer, it is necessary to decide how the various matrices will be spread among the processors. The two simplest organizations are described below, and are referred to as the "column" and "row" versions of the algorithm. Let m be the block size (equal to the number of processors). In storing matrices, we are referring to the larger $n \times m$ matrices; it will be assumed that identical copies of the smaller $m \times m$ matrices are stored on each processor so that there are no communication costs associated with using them. (Note that there may be communication costs involved in forming them.) If m is large, then it may be worthwhile to distribute the smaller matrices over the parallel computer as well. The arithmetic costs are not affected by these choices.

If the matrix-vector product is computed via finite differencing (2.2), then to compute the products Gv_i it is necessary that the vector v_i be stored on a single processor (say, the ith). In addition, the result Gv_i will be obtained on the ith processor. This suggests storing all $n \times m$ matrices with one column per processor. This approach simplifies the computation of the matrix-vector products, but it increases the communication costs for the rest of the algorithm [16]. In addition, if the block size decreases (say, due to loss of rank), some processors will become idle.

An alternative is to store the $n \times m$ matrices by rows, roughly n/m rows per processor (if m does not exactly divide n, then some processors will have one more row

than others). This increases the communication costs for the matrix-vector product (collecting the vector v_i on processor i, and then spreading the result Gv_i), but the communication costs for the rest of the algorithm are reduced. These two approaches are tested in §4.

With the row organization, the algorithm has near perfect load balancing. Every time a function evaluation is required, all processors will perform a function evaluation simultaneously. Also, the linear algebra operations will be spread evenly over the processors (although there will be slight differences if m does not exactly divide n). The linear algebra work load will remain balanced even if there is a rank drop within the inner algorithm.

Even so, under certain circumstances it may be sensible to choose the block size less than the number of processors (see §1). Our software allows this option, but it has not been used in the tests in this paper. Even with the reduced block size, the linear algebra work load will remain balanced.

3.2. Preconditioning. A truncated-Newton algorithm without preconditioning is not a competitive algorithm. The goal of the preconditioner is to improve the performance of the inner algorithm, reducing the number of inner iterations. In selecting a preconditioner, it is useful to note the convergence properties of the inner algorithm. The (scalar) linear conjugate-gradient method (in exact arithmetic) converges in a finite number of iterations equal to the number of distinct eigenvalues of the Hessian. In addition, before obtaining the exact solution, the algorithm displays a linear rate of convergence with constant $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$, where κ is the condition number of G (in the 2-norm) [9]. Hence a preconditioner should try to reduce the number of distinct eigenvalues of the Hessian, reduce the condition number of the Hessian, or both.

An effective automatic preconditioner for the scalar method is described in [14]; it is a limited-memory quasi-Newton method. This formula in turn is scaled by a diagonal approximation to the Hessian obtained by BFGS updating using the matrix-vector products from the inner iteration. Because of the initialization scheme used (see below), and because of the communication costs required to implement it, the limited-memory quasi-Newton update has been dropped, and only a diagonal scaling is used here.

We tested three related diagonal scalings. The first is the diagonal of the Hessian computed via finite differencing. This will not normally be practical unless these Hessian elements can be obtained at low cost, but it provides an "upper bound" on the performance of more practical techniques. The other two scalings are obtained from the formula

$$D_+ = D - \frac{(Dv)(Dv)^T}{v^T D v} + \frac{rr^T}{v^T r},$$

where D is the current diagonal scaling, D_+ is the new scaling, and the vectors v and $r = Gv$ are a matrix-vector product pair from the inner algorithm. This is the diagonal part of a BFGS quasi-Newton update. The two scalings differ only in the computation of the denominator term $v^T D v$. One uses the formula as above, and so the updates to the diagonal scaling must occur sequentially. The other computes $v^T D v$ using the value of D at the beginning of the current set of updates, and so all the updates can be performed in parallel. This second version also reduces the communication costs for performing the updates. (Note that these diagonal scalings differ from that used in [15].)

Although there are theoretical justifications for using the diagonal of the Hessian

as a diagonal scaling [28], there is no guarantee that the exact diagonal of the Hessian will even *reduce* the condition number, even if the Hessian is diagonally dominant [21]. However, extensive numerical testing indicates that it is one of the best general-purpose preconditioning strategies available.

An alternative preconditioning strategy based on the eigenvalues of the Hessian matrix is examined in [21].

3.3. Initialization. In the scalar Lanczos method, one initial vector is required. To make the algorithm equivalent to the conjugate-gradient method, the right-hand side g is used. In the block case m initial vectors are needed, and one of them is chosen as the right-hand side g. The others need only be chosen to form a linearly independent set.

Various simple choices for these vectors are mentioned in [18]. A more theoretically satisfying (and more effective) choice can be based on the idea of a limited-memory quasi-Newton method. In such a method, the search direction is a linear combination of old gradients and search directions g_k, p_{k-1}, g_{k-1}, $\cdots$. The number of terms used depends on the storage made available. The specific linear combination used is defined by quasi-Newton update formulas, usually the BFGS formula.

In the block-Lanczos method, the search direction obtained at the end of the first inner iteration is a linear combination of the set of initial vectors used, chosen to minimize the value of the quadratic model. Our initialization scheme uses old gradients and search directions as the initial vectors for the method

$$V_{(1)} = [g(x_k)|p_{k-1}|g(x_{k-1})|p_{k-2}|\cdots].$$

Then the first approximate search direction will be an "optimal" linear combination of the same vectors used to obtain the limited-memory quasi-Newton direction. Since the limited-memory direction can be of high quality [15], it is expected that this initialization scheme would improve the performance of the overall truncated-Newton algorithm. This scheme increases the storage requirements for the algorithm by one n-vector per processor.

Some details need to be clarified. First, at the early outer iterations, there will not be enough old gradients and search directions to provide a complete set of initial vectors. In this case, random vectors are used for the unavailable vectors. Second, the initial vectors may not form a linearly independent set; random vectors are used as needed.

This technique overlaps with the preconditioning ideas discussed above. The old search direction p_{k-1} satisfies $G(x_{k-1})p_{k-1} \approx -g(x_{k-1})$, and so, as the solution to the minimization problem is approached, we can expect that

$$p_{k-1} \approx -G(x_{k-1})^{-1}g(x_{k-1}) \approx -G(x_k)^{-1}g(x_{k-1}).$$

Thus, p_{k-1} will approximate the outcome of one iteration of the inverse power method for finding eigenvalues. This will bias p_{k-1} in the directions of the eigenvectors corresponding to the small eigenvalues of $G(x_k)$. These eigenvectors will then be well represented in the block Krylov subspace generated by the block-Lanczos method. This property can accelerate the convergence of the inner algorithm.

3.4. Stopping the inner iteration. The initial convergence results for truncated-Newton methods [5] assess the search direction in terms of the scaled norm of the residual of (2.1), $r_i \equiv \|Gp_i + g\|_2 / \|g\|_2$, where i is the index for the inner iteration. If

the search direction is accepted when r_i is appropriately small, any desired asymptotic rate of convergence between linear and quadratic can be achieved.

The residual, however, may be an arbitrarily poor predictor of the quality of a search direction [19]. More reliable acceptance tests can be designed based on the value of the quadratic function $Q(p)$. The test we use terminates the inner iteration if

$$\frac{i[Q(p_i) - Q(p_{i-1})]}{Q(p_i)} \leq \epsilon,$$

where ϵ is a convergence tolerance (various values of ϵ are tested in the next section). Here, i is the number of the inner iteration, and p_i is the search direction obtained at the ith inner iteration. Justification for this test can be found in [19].

The inner iteration is also terminated if indefiniteness in the Hessian is detected. In addition, if the line search at the previous outer iteration took a step $\alpha \neq 1$, then only one inner iteration is allowed. The (block) conjugate-gradient iteration used here produces well-scaled search directions [14], and hence ensures that a step of $\alpha = 1$ should be acceptable if the quadratic approximation $Q(p)$ to $f(x)$ is accurate. Thus, if $\alpha \neq 1$ we conclude that the quadratic approximation is not accurate, and hence that there is little practical or theoretical justification for finding an exact minimizer of $Q(p)$.

3.5. A parallel line search. The complete details of the line search are given in [17]; we give a brief description here. At each iteration of the search, a set of trial steplengths $\{\alpha_i\}_{i=1}^m$ are chosen. (Choices for this set are examined below.) Each processor is assigned a steplength, and evaluates $f(x + \alpha_i p)$ and $g(x + \alpha_i p)$. The value of α that produces the minimal value of f is chosen as the steplength, among those α that satisfy an Armijo-type acceptance test $f(x + \alpha p) \leq f(x) + \eta \alpha g(x)$, with $\eta = 0.2$. If no acceptable steplength is found, a new set of steplengths is constructed, with smaller steplengths than the first. If the largest α is the best one, a new set of steplengths is used, but with larger steplengths instead. A line search based on the Wolfe conditions [12] could also be used.

The choice of the initial steplengths in the line search was varied. The default option used above chose the largest steplength to be m, the number of processors. For comparison, we ran variants with the largest steplength as $2m$, and as $2^{m/2-1}$, with $m = 16$. The default option was slightly better than the others. We chose to use this choice in the later tests, in part based on its performance here, but also to avoid the risk of overflow on functions that increase rapidly near the initial guess x_0.

Alternative line search algorithms, including a nonmonotone line search [11], are discussed in [21].

3.6. Summary of algorithmic costs. We list here the costs of the block truncated-Newton method, using an automatic preconditioner and the most elaborate initialization scheme. All are computed "per processor." If m is the number of processors (equal to the block size), n is the number of variables, and $r = n/m$, then the storage required is

$$S = 12n + 4m^2 + 12m + 2(r + 1).$$

All but $3m$ of this is storage for real numbers; $3m$ integer storage locations are required.

Measuring the arithmetic and communication costs of the algorithm is complicated by the fact that the algorithm has both an inner and outer iteration, and that

the line search can have variable costs. At each outer iteration, the fixed costs for arithmetic are $A_1 = 2n + 8r + m^2 + m$ (counting all arithmetic operations equally), and $C_1 = n + 3r + 2$ numbers must be communicated (in five separate messages). At the inner iteration the arithmetic costs are

$$A_2 = 6mn + 22n + m^3 + 11m^2 + 7m + 2r + [1 \text{ evaluation of } f(x)].$$

The communication costs (in $2m + 2$ messages) are $C_2 = 2n + 2m^2 + m + 1$. In the line search, each iteration has arithmetic costs of $A_3 = 2n + [1 \text{ evaluation of } f(x)]$ and communication costs (in one message) of $C_3 = 3$.

Typically, the line search will only require one iteration to find an acceptable steplength. If we assume two inner iterations to compute a search direction, then a "typical" outer iteration will have arithmetic costs of

$$A = A_1 + 2A_2 + A_3 = 48n + 12mn + 2m^3 + 23m^2 + 12r + 15m + [3 \text{ evaluations of } f(x)]$$

and communication costs (in $4m + 10$ messages) of

$$C = C_1 + 2C_2 + C_3 = 5n + 3r + 4m^2 + 2m + 7.$$

Many of the low-level operations in this algorithm are standard vector operations (inner products, sums of vectors, etc.). On the Intel hypercube that we are using in our tests, these operations can be performed using built-in subroutines, and we have done this wherever possible. In addition, our hypercube is equipped with vector processors that will execute these operations at even greater speed, if the number of variables is sufficiently large (there is some overhead associated with the use of the vector processors). Hence the algorithm is not only suitable for a parallel computer, but is also rich in lower-level vector operations, making it especially well suited for the Intel vector hypercube, as well as for computers, such as the Alliant, which have both parallel and vector capabilities.

4. Numerical results. The problems used in our tests are not new, to facilitate comparison with other algorithms. In particular, we have retained the numbering of the problems used in [15]. They are described in Table 1, listing their names and the number of variables selected; problems 54 and 55 are from [4], and the rest are from [15].

The results of the tests are given in Tables 2–6. There "it" is the outer iteration count, "f/g" is the total number of function/gradient evaluations used per processor (this reflects both line search and inner iteration costs), and "Time" is the time required to solve the problem in seconds. The runs were made on an Intel iPSC/2 hypercube computer with 16 processors, each with a vector coprocessor. The vector coprocessors were not used on all runs due to hardware breakdowns (this is indicated in the table captions). The computer was programmed using Fortran in double precision (about 16 decimal digits). An upper bound of 500 outer iterations was imposed, except for the results in Table 5, where 1000 outer iterations were allowed.

Initial tests were used to determine a "definitive" version of the block truncated-Newton code, using a subset of the test problems. The later tests compare this final version with other optimization software. In the initial tests, the algorithm was stopped when

$$\|g(x_k)\|_\infty \leq 10^2 \sqrt{\epsilon}(1 + |f(x_k)|),$$

TABLE 1
List of test problems.

Problem	Name	n
1	Calculus of variations 1	100, 200, 1000
2	Calculus of variations 2	100, 200, 1000
3	Calculus of variations 3	100, 200, 1000
6	Generalized Rosenbrock	100, 500, 1000
8	Penalty 1	100, 1000
9	Penalty 2	100
10	Penalty 3	100, 1000
28	Extended Powell singular	100, 1000
29	Variably dimensioned	100, 500
31	Brown almost linear	100, 200
38	Tridiagonal 1	100, 1000
39	Linear minimal surface	121, 961
40	Boundary-value problem	100, 1000
41	Broyden tridiagonal nonlinear	100, 1000
42	Extended ENGVL1	100, 1000
43	Ext. Freudenstein and Roth	100, 200, 1000
45	Wrong extended Wood	100, 1000
46^1	Matrix square root ($ns = 1$)	100, 1000
46^2	Matrix square root ($ns = 2$)	100, 1000
47	Sparse matrix square root	100, 1000
48	Extended Rosenbrock	100, 200, 1000
49	Extended Powell	100, 200, 1000
50	Tridiagonal 2	100, 1000
51	Trigonometric	100, 1000
52	Penalty 1 (2nd version)	100, 1000
53	INRIA ults0	403
54	Toint 61	200, 1000
55	Toint 62	200, 1000

where $\epsilon \approx 2.2 \times 10^{-16}$ is the machine epsilon for the hypercube. Complete results for these initial tests are given in [21]; only summary information is given here.

First, the row and column versions were compared (these differ only in implementation). To ensure that both versions performed the same arithmetic computations, the algorithm was stopped after 10 outer iterations, and only 1 inner iteration was allowed. On problem 1 with $n = 50$, the row version was more than twice as fast, confirming our earlier comments. The row version was used in all later tests. The remaining preliminary tests were run on 14 problems (functions 1, 2, 3, 6, 8, 28, 31, 38, 41, 43, 45, 46^2, 47, and 51), all with $n = 100$.

Four preconditioners were compared: PC0 (no preconditioning), PC1 (exact diagonal of Hessian), PC2 (BFGS diagonal scaling), and PC3 (approximate BFGS diagonal scaling, with lower communication costs). PC1 is not practical, but was included to indicate "ideal" performance for a diagonal preconditioner. (Computing the diagonal entries of the Hessian requires almost as much computation as computing the full Hessian.) On the 14 problems, PC0 took 1112 seconds, PC1 took 401, PC2 took 393,

TABLE 2

Comparison with [15] (*smaller problems*). *Failures were counted as* 1000 *iterations and* 2000 *function evalutions.*

F	n	BTN		LBFGS		TN	
		it	f/g	it	f/g	it	f/g
1	100	9	52	*F*		28	466
2	100	7	39	1605	1669	27	242
3	100	8	50	3095	3216	45	325
6	100	90	223	257	291	78	683
8	100	11	30	30	36	4	26
9	100	6	15	21	23	8	48
10	100	14	50	82	90	20	107
28	100	27	104	57	67	14	70
29	100	7	18	36	37	9	42
31	100	5	18	24	27	14	101
38	100	11	43	120	128	18	77
39	121	20	72	66	70	18	187
40	100	11	63	2219	2296	55	2091
41	100	16	45	28	31	14	77
42	1000	9	26	15	17	15	75
43	100	6	20	19	21	10	38
45	100	10	35	48	56	14	59
46[1]	100	19	83	362	377	30	339
46[2]	100	25	112	438	453	35	556
47	100	15	59	87	95	17	94
48	1000	39	144	38	49	16	79
49	100	26	102	57	67	14	70
50	100	14	53	129	133	17	78
51	100	25	64	53	60	28	233
52	1000	3	10	5	6	3	10
Totals		433	1530	9891	11315	551	6173

and PC3 took 373. (However, PC2 required only 1289 function evaluations to PC3's 1345.) Based on these tests, we felt that PC3 was the best preconditioner, and it has been used in the subsequent tests. This decision was influenced by the relatively high communication costs on the hypercube computer. If communication were faster (or on a shared-memory machine), then PC2 would be chosen.

Next we compared the initialization schemes: INITV1 (random vectors), INITV2 (old search direction plus random vectors), and INITV3 (quasi-Newton initialization). On the 14 problems, INITV1 took 373 seconds, INITV2 took 326, and INITV3 took 238 (the function-evaluation counts were proportional to the times). INITV3 was the best strategy and it was used in subsequent tests. Note that this scheme requires no extra computations, yet reduces the costs of the algorithm by one-third over random initial vectors, and for problem 6, by a factor of more than three.

Other experiments did not identify major improvements in the algorithm; we briefly summarize here the tests made in [21]. First, the inner iteration was scaled by

TABLE 2A

Comparison with [15] (*larger problems*). *Runs marked with* * *were made with convergence tolerance* 10^{-5}, *as in* [15]. *Failures were counted as* 1000 *iterations and* 2000 *function evalutions.*

F	n	BTN		LBFGS		TN	
		it	f/g	it	f/g	it	f/g
1	200	25	185	*F*		38	929
2	200	10	67	1734	1785	37	456
3	200	16	133	7248	7482	76	599
6	500	380	924	1054	1177	356	3446
8	1000	7	18	30	34	12	58
10*	1000	14	52	103	114	30	200
28	1000	41	204	54	61	15	75
29*	500	8	20	48	49	12	54
31	200	5	16	3	4	4	20
38	1000	18	73	405	423	33	208
39	961	41	196	165	172	27	387
43	1000	6	19	16	20	15	75
47	1000	26	161	145	157	22	160
49	1000	41	204	54	61	14	68
50	1000	24	111	457	476	30	210
51*	1000	24	62	46	57	35	370
53	403	13	61	57	63	14	100
Totals		699	2506	12619	14135	770	7415

applying the block algorithm to the linear system

$$Gp = -g/\|g\|_\infty$$

and then multiplying the search direction by $\|g\|_\infty$ before the line search. As the solution is approached, $g \to 0$, and this can lead to scaling problems. This scaling slightly improved performance, and hence this change was retained.

Second, the algorithm was modified so that the inner iteration was terminated if a loss of rank in the Lanczos vectors was detected. (A loss of rank leads to a loss of efficiency in the inner iteration.) However, this idea was not effective and was abandoned.

Third, the inner convergence tolerance was varied. This had little effect overall (although it did influence individual problems). The later tests used the tolerance $\epsilon = .5$.

Fourth, we compared the resulting algorithm with a simplified method where only one inner iteration is allowed. With 16 processors, as well as the preconditioning and initialization schemes, we thought that such an algorithm (with its much-reduced communication costs) might be effective. It was not. This suggests that even though the better algorithm only uses about 2.5 inner iterations per outer iteration on average, occasional use of more inner iterations is important for achieving good performance in general.

Based on these tests, we settled on the following block truncated-Newton method: row organization, preconditioned by an approximate BFGS diagonal scaling and initialized with a quasi-Newton–based scheme, with the inner iteration scaled by the

TABLE 2B

Comparison with [15] *(harder problems). Failures were counted as* 1000 *iterations and* 2000 *function evalutions. The row marked "*BTN *speedup" indicates the speedup of* BTN *over the scalar routines, as measured by the number of function/gradient evaluations required per processor.*

	BTN	TN	LBFGS
it	746	988	> 21241
f/g	2489	11740	> 24048
wins	14	0	3
failures	0	0	2
BTN speedup	–	4.7	9.7

TABLE 3

Comparison of BTN *with parallel algorithms from* [4]. *Eight processors were used. The convergence test is as in* [4]*; the runs in that paper were performed on a different computer.*

F	n	BTN		PVMP		FDCP/FDNCP	
		it	f/g	it	f/g	it	f/g
43	200	6	20	5	131	5	131
48	200	48	169	895	45646	598	7983
49	200	28	137	15	391	17	443
54	200	10	29	9	235	10	261
55	200	18	97	13	339	13	339

norm of the gradient, with the inner iteration terminated based on the value of the quadratic model with tolerance 0.5, and with a parallel line search with maximum initial step equal to m (the number of processors). This algorithm will be labelled BTN in the tests below.

We then compared the algorithm with the (scalar) methods tested in [15]. These are a limited-memory quasi-Newton method called LBFGS (written by Nocedal) and a truncated-Newton method called TN (written by Nash). These were chosen because they are considered to be good scalar methods, among the most effective general-purpose codes for large-scale problems. The numbering of the problems in that paper is the same as that used here, as is the convergence test:

$$\|g(x_k)\|_\infty \leq \epsilon(1 + |f(x_k)|),$$

where $\epsilon = 10^{-6}$ for all problems except numbers 10 ($n = 1000$), 29 ($n = 500$), and 51 ($n = 1000$), where $\epsilon = 10^{-5}$. The results are presented in Tables 2 and 2A; the division into "smaller" and "larger" problems matches that in [15]. Due to memory limitations on our parallel computer, we were not able to run the problems with 10,000 variables.

Based on initial test runs with the expanded test set, two additional changes were made to the block truncated-Newton method. First, it was observed that the performance on a few of the problems (most notably problems 1 and 29) was sensitive to the tolerance used in the rank test within the inner algorithm. It had been set

at 10^{-12}, but was changed to 10^{-9} based on selective computational testing. As has been noted in [6], it is not possible to set rank tolerances in such a way as to solve all problems well.

Second, it was noticed that on some problems there were dramatic changes in the value of the objective function between the initial and final points (most notably problem 29). In such cases the automatic preconditioner may slow the algorithm if it cannot change rapidly enough. To compensate for this problem, the diagonal scaling matrix was reset to the identity matrix whenever $|f(x_k)| \leq 10^{-5}|\bar{f}|$, where $\bar{f}$ was the value of the objective function the last time the scaling matrix was reset. Other resetting schemes were also tried, using different tolerances in the condition above, and based on the iteration count (allowing at most 10 iterations between resets, for example), but these did not further improve the performance of the algorithm.

What can be concluded from Tables 2 and 2A? First, the parallel algorithm BTN does not much improve on the iteration count of the scalar algorithm TN. This suggests that both algorithms are getting a Newton-like direction. However, BTN reduces the overall number of gradient evaluations by a factor of 3–4, indicating that it is obtaining a comparable direction at lower cost (per processor). Based on the totals, BTN is a considerable improvement over LBFGS, although it should be noted that the totals are strongly influenced by problems 1, 2, 3, and 40. BTN appears to be robust. It solves all the problems, and on 26 of the 42 runs uses fewer gradient evaluations (per processor) than *either* of the other two algorithms.

The performance of BTN is especially good on problems 1, 2, 3, and 40. On these problems it achieves speedups of from 5–33 over the *better* of the two scalar algorithms (based on gradient evaluations per processor). This is gratifying since these problems were among the most difficult for the scalar algorithms TN and LBFGS.

In general, BTN performs well on the harder test problems. In Table 2B we extract results for the 17 test problems where *either* of the scalar algorithms required more than 200 gradient evaluations. Over this subset of the problems, BTN achieves a speedup of about 10 over LBFGS and about 5 over TN. Because these are considered good scalar algorithms, and because this is a varied set of test problems, we think that this is a strong indication of the effective performance of BTN on general problems. (Note that TN is not just BTN with the block size set equal to 1. TN has a more effective preconditioner and a more sophisticated line search than BTN.)

On some problems, however, BTN is not impressive. We have examined the worst cases to try to understand why. Problems 28, 48, and 49 consist of identical copies of small problems having 2–4 variables. As a result, the algorithm will behave much as it would if it were solving the small 2–4 variable problem, and increasing the block size will not offer advantages. In addition, even though the Hessian matrix will be block diagonal with identical blocks, the automatic preconditioning strategies can destroy this structure, hence making the problem harder to solve.

On problems 6 and 47, the Hessian at the initial point has negative eigenvalues (for problem 47, half the eigenvalues are negative at the initial point). If the Hessian has negative eigenvalues, the quadratic model of the nonlinear function does not have a minimum, and hence expending considerable effort to approximate the Newton direction, as in BTN, may be wasteful. This may explain BTN's lackluster performance on these two problems.

The parallel line search appears to be effective. Ideally, there will be one function evaluation in the line search per outer iteration. For these test problems, each processor evaluates $f(x)$ an average of 1.0532 times per outer iteration.

Finally, BTN almost always produces low iteration and line search counts (the

TABLE 4

Using BTN *with varying numbers of processors p (with the block size equal to the number of processors). Speedups are computed based on numbers of function/gradient evaluations ("f/g") as well as on time ("time") measured in seconds.*

F	n	p	it	f/g	Speedup	time	Speedup
1	100	1	32	732	–	105	–
		2	27	418	1.8	64	1.6
		4	24	254	2.9	45	2.3
		8	20	156	4.7	36	2.9
		16	9	52	14.1	20	5.3
40	100	1	221	9076	–	692	–
		2	204	5098	1.8	441	1.6
		4	134	1678	5.4	186	3.7
		8	38	307	29.6	50	13.8
		16	11	63	144.1	19	36.4
46[1]	100	1	28	433	–	56	–
		2	25	325	1.3	46	1.2
		4	29	303	1.4	50	1.1
		8	21	125	3.5	26	2.2
		16	19	83	5.2	26	2.2
49	100	1	21	94	–	5	–
		2	39	210	0.4	11	0.5
		4	27	144	0.7	12	0.4
		8	26	118	0.8	15	0.3
		16	26	102	0.9	25	0.2

sole exception being problem 6 for $n = 500$). In circumstances where the matrix-vector product required by the inner algorithm could be produced easily, it would be a good algorithm to use. This would happen for quadratic problems, problems with sparse Hessians, and other special cases.

Although there has been considerable research on the topic of parallel optimization, we are not aware of any other general-purpose software for solving large problems. We do not consider it fair to compare with algorithms based on parallel finite differencing for gradient values, since we assume that the exact gradient values can be computed. The only comparison we have been able to make, and it is not perfect, is with the results in [4]. The authors of that paper worked on an 8-processor shared-memory computer, and were not immediately concerned with producing a competitive general-purpose algorithm. The algorithms in that paper were a parallel quasi-Newton method (Straeter's algorithm), and a finite-difference Newton-type method similar to the truncated-Newton algorithm used here.

To make the comparison, we have modified our outer convergence test to match that in [4]. The test problems used are (our numbering/their numbering): 43/33, 48/10, 49/19, 54/61, 55/62, all with $n = 200$. We record iteration counts ("it") and gradient evaluations per processor ("f/g"). Our algorithm was run using an 8-processor hypercube, and the results are recorded in Table 3. The algorithms PVMP and FDCP/FDNCP are described in [4], and the results were taken from Tables 2 and 3 in that paper. As can be seen from the table, BTN is considerably more efficient

TABLE 5

Some large problems not previously tested. Using algorithm BTN *but allowing* 50 *inner iterations per outer iteration. The runs were made using the vector processors.*

F	n	BTN		
		it	f/g	time
1	1000	138	2695	5371
2	1000	26	486	948
3	1000	105	2067	6306
6	1000	717	1712	883
40	1000	8	85	85
41	1000	19	59	35
45	1000	13	56	41
46^1	1000	51	1278	4885
46^2	1000	51	1196	4556
54	1000	8	24	18
55	1000	22	105	79

than the other two algorithms.

In Table 4 we illustrate the effect of varying block size on the performance of BTN, using a subset of the test problems. The number of inner iterations allowed per outer iteration was varied with the block size: 20 inner iterations for block size 16, 40 for block size 8, and 80 for block sizes 4, 2, and 1. The vector processors were not used on these runs. (Further results can be found in [21].)

Note that as the block size changes, the algorithm also changes. Hence it is possible to get speedups greater than 16 (problem 40) as well as deterioration in performance (problem 49). On the problems that are most difficult using 1 processor (1, 2, 3, 6, 40, and 46^1), increasing the block size led to a reduction in the number of gradient evaluations per processor. Reducing the number of gradient evaluations does not always reduce the overall time to solve the problem, since increasing the block size also increases the amount of communication in the algorithm (see problems 6 and 51). On a shared-memory machine we would observe the same number of gradient evaluations, but would expect lower times since there would not be associated communication costs (although there might be memory bottlenecks).

As can be seen from Tables 2 and 2A, BTN with block size 16 can solve almost all the test problems quickly. Since we expect that future research will improve on this performance, it will be useful to have some harder problems available for comparative testing. To this end we include in Table 5 the results of solving some additional problems using BTN.

Finally, we compare BTN with a modified Newton method. To simulate Newton's method, we ran algorithm TN with the inner convergence test replaced by

$$\frac{\|Gp + g\|_2}{\|g\|_2} \leq 10^{-8},$$

and with an upper limit of 500 inner iterations imposed. In Table 6 we record the results for the problems in Tables 2 and 2A, *with the costs of the inner iteration ignored.* As can be seen, Newton's method only reduces the iteration count slightly. BTN has lower gradient costs (per processor) because of the parallel line search.

TABLE 6

Comparison with (simulated) Newton's method. All inner iteration costs ignored. Only algorithm BTN *uses a parallel line search.*

TN		BTN		Newton	
it	f/g	it	f/g	it	f/g
1321	2889	1132	1240	861	2861

More can be said about this. Suppose that we had an "ideal" parallel Newton method, where the Hessian matrix could be computed and the Newton equations solved for a search direction at a cost of one gradient evaluation per processor. Suppose also that our parallel line search was used to compute the steplength. Hence, this "ideal" Newton method would require about 2 gradient evaluations per major iteration. Note that, on average, BTN only requires about 2.5 inner iterations (and hence about 2.5 gradient evaluations) to determine a search direction. Thus BTN needs, on average, 3.5 gradient evaluations per major iteration, and has only a slightly higher iteration count. Hence BTN would only be about twice as expensive as this "ideal" Newton method on this test set. Thus BTN, a general-purpose parallel optimization algorithm suitable for large-scale optimization, can approach the performance of Newton's method, without the high costs associated with Newton's method.

Acknowledgments. We would like to thank David Scott for his helpful comments on using the hypercube computer. The computer we used, an Intel iPSC/2 hypercube computer, belongs to the Center for Computational Statistics at George Mason University, and was obtained with the aid of U.S. Army Office of Research contract DAAL03–87–K–0087.

REFERENCES

[1] R.H. BYRD, R.B. SCHNABEL, AND G.A. SHULTZ, *Using parallel function evaluations to improve Hessian approximations for unconstrained optimization*, Tech. Report CU–CS–361–87, Department of Computer Science, University of Colorado, Boulder, CO, 1987.

[2] ———, *Parallel quasi-Newton methods for unconstrained optimization*, Tech. Report CU–CS–396–88, Department of Computer Science, University of Colorado, Boulder, CO, 1988.

[3] P. CONCUS, G. H. GOLUB, AND D. P. O'LEARY, *A generalized conjugate-gradient method for the numerical solution of elliptic partial differential equations*, in Sparse Matrix Computations, J. Bunch and D. Rose, eds., Academic Press, New York, 1976, pp. 309–332.

[4] M. DAYDE, M. LESCRENIER, AND PH.L. TOINT, *A comparison between Straeter's parallel variable metric algorithm and parallel discrete Newton's methods*, Report 89/16, Facultés Universitaires de Namur, Namur, Belgium, 1989.

[5] R.S. DEMBO AND T. STEIHAUG, *Truncated-Newton algorithms for large-scale unconstrained optimization*, Math. Programming, 26 (1983), pp. 190–212.

[6] C. FRALEY, *Computational behavior of Gauss–Newton methods*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 515–532.

[7] P.E. GILL AND W. MURRAY, *Safeguarded steplength algorithms for optimization using descent methods*, Report NAC 37, National Physical Laboratory, Teddington, England, 1974.

[8] ———, *Newton-type methods for unconstrained and linearly constrained optimization*, Math. Programming, 28 (1974), pp. 311–350.

[9] G.H. GOLUB AND C.F. VAN LOAN, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 1989.

[10] A. GRIEWANK *On automatic differentiation*, Mathematical Programming, M. Iri and K. Tanabe, eds., Kluwer Academic Publishers, Tokyo, 1989, pp. 83–107.

[11] L. GRIPPO, F. LAMAPARIELLO, AND S. LUCIDI, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707–716.

[12] D.G. LUENBERGER, *Introduction to Linear and Nonlinear Programming*, Addison–Wesley, Reading, MA, 1973.

[13] S.G. NASH, *Newton-like minimization via the Lanczos method*, SIAM J. Numer. Anal., 21 (1984), pp. 770–788.

[14] ———, *Preconditioning of truncated-Newton methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 599–616.

[15] S.G. NASH AND J. NOCEDAL, *A numerical study of the limited memory* BFGS *method and the truncated-Newton method for large scale optimization*, Report NAM 02, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL, 1989.

[16] S.G. NASH AND A. SOFER, *Parallel optimization via the block Lanczos method*, in Computer Science and Statistics: Proceedings of the 20th Symposium on the Interface, E. Wegman, D. Gantz, and J. Miller, eds., American Statistical Association, Alexandria, VA, 1989, pp. 209–213.

[17] ———, *A parallel line search for Newton-type methods*, in Computer Science and Statistics: Proceedings of the 21st Symposium on the Interface, K. Berk and L. Malone, eds., American Statistical Association, Alexandria, VA, 1989, pp. 134–137.

[18] ———, *Block truncated-Newton methods for parallel optimization*, Math. Programming, 45 (1989), pp. 529–546.

[19] ———, *Assessing a search direction within a truncated-Newton method*, Oper. Res. Lett., 9 (1990), pp. 219–221.

[20] ———, BTN: *Software for parallel unconstrained optimization*, Report 64, Center for Computational Statistics and Probability, George Mason University, Fairfax, VA, 1990.

[21] ———, *A general-purpose parallel algorithm for unconstrained optimization*, Report 63, Center for Computational Statistics and Probability, George Mason University, Fairfax, VA, 1990.

[22] D.P. O'LEARY, *The block conjugate-gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[23] ———, *A discrete Newton algorithm for minimizing a function of many variables*, Math. Programming, 23 (1983), pp. 20–33.

[24] ———, *Parallel implementation of the block conjugate gradient algorithm*, Parallel Comput., 5 (1987), pp. 127–139.

[25] J.M. ORTEGA AND W.C. RHEINBOLDT, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, London, New York, 1970.

[26] C.C. PAIGE AND M.A. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629.

[27] P.M. PARDALOS, A.T. PHILLIPS, AND J.B. ROSEN, *Topics in parallel computing in mathematical programming*, Report CS–90–22, Department of Computer Science, The Pennsylvania State University, University Park, PA, 1990.

[28] A. VAN DER SLUIS, *Condition numbers and equilibration of matrices*, Numer. Math., 14 (1979), pp. 14–23.

[29] S.A. ZENIOS, *Parallel numerical optimization: Current status and an annotated bibliography*, ORSA J. Comput., 1 (1989), pp. 20–43.

[30] S.A. ZENIOS AND J.M. MULVEY, *Nonlinear network programming on vector supercomputers: A study on the Cray* X–MP, Oper. Res., 34 (1986), pp. 667–682.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 548–564, November 1991

008

ACCELERATION AND PARALLELIZATION OF THE PATH-FOLLOWING INTERIOR POINT METHOD FOR A LINEARLY CONSTRAINED CONVEX QUADRATIC PROBLEM*

Y. NESTEROV† AND A. NEMIROVSKY†

Abstract. In this paper, the strategies for acceleration of the path-following polynomial time interior point method for linear and linearly constrained quadratic programming problems are studied. These strategies are based on (i) exploiting the results of computations done at the previous iterations (Karmarkar's acceleration scheme and a scheme based on the preconditioned conjugate gradient method); (ii) implementation of "fast" linear algebra routines; (iii) parallel computations.

Key words. interior point methods, polynomial time algorithms, parallel computations

AMS (MOS) subject classifications. 90C05, 90C20

1. Introduction.

In this paper we consider the problem

$$\begin{aligned} &\psi(x) \equiv \tfrac{1}{2}x^T\Theta x - a^T x \to \min \,|\, x \in R^n, \\ &A^T x \leqq b, \end{aligned} \tag{1}$$

where Θ is a symmetric positive semidefinite $n \times n$ matrix, A is an $n \times m$ matrix, and $b \in \mathbb{R}^m$. In other words, we deal with a linearly constrained convex quadratic programming problem. Note that the results which follow do not change if we replace such a problem with a linear programming problem, i.e., if we set $\Theta = 0$.

From now on, we set

$$G = \{x \in R^n \,|\, A^T x \leqq b\}. \tag{2}$$

We assume that G is a bounded set with a nonempty interior (hence $m > n$). Without loss of generality, we suppose that A does not contain zero columns. Then

$$G' \equiv \operatorname{int} G = \{x \in R^n \,|\, A^T x < b\}. \tag{3}$$

At least from the theoretical viewpoint, the best known methods for (1) are the polynomial time interior point methods. These methods originate from the seminal papers of Karmarkar [Ka 1984] and Renegar [Re 1988] devoted to LP problems; important contributions to the area were made by many other researchers. In particular, extensions of interior point methods for LP to the linearly constrained quadratic problems were developed by Goldfarb and Lui [GL 1988]; Kojima, Mizuno, and Yoshise [KMY 1988], [KMY 1989]; Monteiro and Adler [MA 1989a], [MA 1989b]; and Nesterov [Ne 1988a], [Ne 1988b].

The preliminary formulation of the question studied in this paper is how one can reduce the time complexity of these methods with the aid of parallel computations. The answer obviously does not depend on the concrete interior point method, and we, for the sake of simplicity, restrict ourselves to the barrier path-following method [Go 1987], [Ne 1988a], [Ne 1988b]. This method is based on the logarithmic barrier

$$F(x) = -\sum_{i=1}^{m} \ln (b_i - (A^T x)_i) \colon G' \to \mathbb{R}. \tag{4}$$

* Received by the editors August 17, 1990; accepted for publication (in revised form) February 22, 1991.
† Central Economical & Mathematical Institute, USSR Academy of Sciences, 32 Krasikova Str., 117418 Moscow, USSR.

The *main stage* of the method is as follows. Consider the family of functions

$$F_t(x) = t\psi(x) + F(x)\colon G' \to \mathbb{R},$$

$t > 0$, and the corresponding trajectory of minimizers

$$x^*(t) = \operatorname{argmin}\{F_t(x) \mid x \in G'\},$$

which converges to the optimal solution of (1). In the barrier method we approximate this trajectory along the sequence $\{t_i = t_0\kappa^i\}$, $\kappa = 1 + \Theta(m^{-1/2})$, for values of the barrier parameter t. The ith approximate solution, x_i, is to be close to the point $x^*(t_i)$, say, in the sense that

$$F_{t_i}(x^i) - F_{t_i}(x^*(t_i)) \leqq \mu, \tag{5}$$

where μ is an appropriate absolute constant. In the above-mentioned papers, it is proved that, in order to maintain (5), it suffices to update x^i with the aid of a single Newton step

$$x^{i+1} = x^i - [\nabla_x^2 F_{t_{i+1}}(x^i)]^{-1} \nabla_x F_{t_{i+1}}(x^i), \tag{6}$$

provided that (5) holds for $i = 0$. The latter condition can be satisfied if x^0 is close enough to the minimizer, $x(F)$, of the barrier over G' and if t_0 is small enough. To approximate $x(F)$, one can also use the path-following technique, but, in a sense, in inverted time. Namely, assume that we are given an initial point, $w \in G$, that is a strictly feasible solution to our problem. Consider the family

$$\bar{F}_t(x) = -tx^T \nabla F(w) + F(x). \tag{7}$$

The corresponding trajectory of minimizers, $x_*(t) = \operatorname{argmin}\{\bar{F}_t(x) \mid x \in G'\}$ passes through w when $t = 1$ and converges to $x(F)$ as $t \to 0$. We can approximate the points $x_*(t_i)$ along the sequence $\{t_i = \kappa^{-i}\}$ with the points x_i defined as

$$x_0 = w; \qquad x_{i+1} = x_i - [\nabla_x^2 \bar{F}_{t_{i+1}}(x_i)]^{-1} \nabla_x \bar{F}_{t_{i+1}}(x_i), \tag{8}$$

which allows us to maintain the relation similar to (5). This *preliminary stage* is terminated when t_i becomes small enough, the final x_i being the desired approximation to the minimizer of the barrier.

In the above-cited papers (see, e.g., [Ne 1988b]) it is proved that the outlined two-stage method finds an ε-solution to our problem, i.e., a feasible solution x_ε such that

$$\psi(x_\varepsilon) - \min_G \psi \leqq \varepsilon \left\{ \max_G \psi - \min_G \psi \right\} \tag{9}$$

in no more than

$$N(\varepsilon) = O(m^{1/2}) \ln (m\delta^{-1}\varepsilon^{-1}) \tag{10}$$

iterations of the preliminary and the main stages. Herein $\delta = 1 - \pi_F(w)$, π_F being the Minkowsky function of G with the pole at the minimizer $x(F)$ of the barrier.

Note that the factor at the logarithmic term is precisely the amount of iterations required to follow the trajectory of minimizers

$$x^\phi(t) = \operatorname{argmin}\{F_t^\phi(x) = t\phi(x) + F(x) \mid x \in G'\} \tag{11}$$

(ϕ is a convex quadratic form) along the segment of the values of the barrier parameter t with the ratio of the endpoints, say, 2, i.e., along the segment $[\tau, 2\tau]$, $\tau > 0$. If we could follow this trajectory with certain time complexity M^*, then the time complexity

of finding an ε-optimal solution to (1) would be

$$M(\varepsilon) = O(M^* \ln (m\delta^{-1}\varepsilon^{-1})). \tag{12}$$

Thus the time complexity of an implementation of the path-following method is defined by the time complexity M^*, at which one can solve the following problem.

BASIC PROBLEM. *Given the segment* $[\tau, 2\tau]$, *a convex quadratic form* ϕ, *and an approximation* x_{in} *to one of the endpoints of the corresponding curve* $x^{\phi}(t)$, $\tau \leqq t \leqq 2\tau$, *say, to the point* $x^{\phi}(\tau)$, *such that*

$$\delta_{\tau}(x_{\text{in}}) \leqq \mu, \tag{13}$$

find an approximation x_{out} *to the other endpoint* (*i.e., to* $x^{\phi}(2\tau)$), *such that*

$$\delta_{2\tau}(x_{\text{out}}) \leqq \mu. \tag{14}$$

Herein μ is a positive *absolute constant* and

$$\delta_t(x) = F_t^{\phi}(x) - F_t^{\phi}(x^{\phi}(t)). \tag{15}$$

In accordance with (12), the p-processor time complexity $\mathfrak{C}_p(\Pi)$ of a procedure Π solving the Basic Problem can be thought of as *the p-processor time complexity per a precision digit of the approximate solution to* (1) *produced by the path-following method based on* Π. Namely, the latter method finds an ε-solution to (1) with the p-processor time complexity not greater than

$$O(\mathfrak{C}_p(\Pi)) \ln (m\delta^{-1}\varepsilon^{-1}).$$

The main result of the path-following method as applied to (1) (see the above-cited papers) is that the Basic Problem can be solved by the following procedure.

BASIC. Let, for a positive m-dimensional vector d,

$$Q(d) = [M(d)]^{-1}, \qquad M(d) = \phi'' + A \operatorname{Diag}\{d\} A^T,$$
$$d^*(t, x) = t^{-1}(b - A^T x)^{-2}. \tag{16}$$

From now on, for m-dimensional vectors $u = (u_1, \cdots, u_m)^T$, $v = (v_1, \cdots, v_m)^T$, uv denotes the componentwise product of u and v, that is, the vector $(u_1 v_1, \cdots, u_m v_m)^T$; notations like u/v, u^{α} ($\alpha \in \mathbb{R}$), etc., are also used in the componentwise meaning.

Note that

$$\nabla_x^2 F_t^{\phi}(x) = tM(d^*(t, x)).$$

BASIC can be described as follows:

BEGIN
$x := x_{in}$;
$t := \tau$;
WHILE $(t < 2\tau)$ **DO**
BEGIN
ITERATION:
UPDATING $t \mapsto t^+$:
$t^+ := \min\{(1 + O(m^{-1/2}))t, 2\tau\}$;
PREPARATION TO UPDATING $x \mapsto x^+$:
$d^+ := d^*(t^+, x)$;
$g^+ := \nabla_x F_{t^+}^{\phi}(x)$;
$Q^+ := Q(d^+)$;
UPDATING $x \mapsto x^+$:
$x^+ := x - (t^+)^{-1} Q^+ g^+$;

RENEWAL:

$$t := t^+;$$
$$x := x^+;$$

END;

$$x_{\text{out}} := x;$$

END.

It is known (see, e.g., [NN 1989]) that the updating rules involved in BASIC ensure the implication

$$\{\delta_t(x) \leqq \mu\} \Rightarrow \{\delta_{t^+}(x) \leqq 2\mu\} \quad \text{and} \quad \{\delta_{t^+}(x^+) \leqq \mu\} \tag{17}$$

for each small enough absolute constant μ, provided that the rate at which the values of t are varied is chosen in an appropriate way (namely, the quantity $O(m^{-1/2})$ is taken equal to $O(1)\mu^{1/2}m^{-1/2}$, where $O(1)$ is a once-and-for-ever-fixed absolute constant). Thus, the above procedure does solve our Basic Problem.

A straightforward single-processor implementation of this procedure on the basis of the standard linear algebra leads to

$$\mathfrak{C}_1(\text{BASIC}) = O(m^{3/2}n^2) \leqq O(m^{3.5}). \tag{18}$$

What are the possibilities of reducing this time complexity? There are at least three of them, namely,

—implementation of "fast" linear algebra algorithms instead of the standard routines;

—recursive computation of the (approximate) inverse Hessians used to perform Newton steps;

—parallelization of computations.

"Fast" linear algebra. It is well known that the product of a pair of $r \times r$ matrices can be computed on a single-processor computer in time of order less than $O(r^3)$; assume that it can be done with the time complexity $O(r^{2+\gamma})$ for certain $\gamma < 1$ (the best known value of γ satisfying this assumption is $0.376\cdots$). It is known that the inversion of an $r \times r$ matrix admits the same complexity bound. Using the corresponding "fast" linear algebra routines in BASIC, we obtain a new procedure BASIC(γ) (so that BASIC = BASIC(1)) such that

$$\mathfrak{C}_1(\text{BASIC}(\gamma)) = O(m^{2.5+\gamma}) \tag{19}$$

(for the sake of simplicity in this preliminary complexity analysis, we set $n = O(m)$, so that the complexity can be expressed in terms of m only).

Recursive computation of (approximate) inverse Hessians. Acceleration based on this possibility originates from Karmarkar [Ka 1984]. In the context of the path-following methods and the standard linear algebra, it was developed in many papers and allowed to reduce the single-processor time complexity of BASIC by a factor $O(m^{1/2})$. Karmarkar's acceleration scheme as applied to BASIC leads to a new procedure KARM with

$$\mathfrak{C}_1(\text{KARM}) = O(m^3). \tag{20}$$

Implementation of "fast" linear algebra routines in the latter procedure (procedure KARM(γ)) results in

$$\mathfrak{C}_1(\text{KARM}(\gamma)) = O(m^{s(\gamma)}), \qquad s(\gamma) = 2.5 + 0.5\gamma. \tag{21}$$

It turns out that in the case of $\gamma < 1$ the latter bound (and, therefore, the Karmarkar acceleration scheme) is not the best one. We managed to develop an essentially new acceleration strategy based on the preconditioned conjugate-gradient method. For the new strategy (procedure CG(γ)), one has

$$\mathfrak{C}_1(\mathrm{CG}(\gamma)) = O(m^{r(\gamma)}), \qquad r(\gamma) = 2.5 + 2\gamma^2/(2+3\gamma-\gamma^2). \tag{22}$$

This strategy is better in order than Karmarkar's for each γ less than 1, say, $r(0.376\cdots) = 2.594\cdots$, $s(0.376\cdots) = 2.688\cdots$.

Parallelization. So far, we have discussed the single-processor computations. What happens when we can use p processors? In this paper we try to answer this question for the simplest SIMD model of computations. More exactly, we are interested in the efficient parallelization only, that is, in the parallelization which allows us to reduce the time complexity of the algorithm by a factor of order p. The question is: for which values of p does the efficient parallelization exist? The best possible answer would be the following: for all p less in order than the single-processor time complexity. It seems to be impossible to obtain such a result for the path-following method, since the essence of the method is a sequential $O(m^{1/2})$-step iterative process. Therefore, parallelization of the method cannot make the time complexity less than $O(m^{1/2})$. Another difficulty is that all efficient (with respect to the single-processor implementation) versions of the path-following method require inverting matrices, and the possibility of optimal parallelization for the latter problem is an open question. Therefore, we are not able to point out *an efficient* parallelization of the method with the time complexity $O(m^{1/2})$. Nevertheless, it turns out that the efficient parallelization does exist for relatively large values of p. Roughly speaking, for both of the accelerations (Karmarkar's strategy and the strategy based on the conjugate-gradient method) the single-processor time complexity can be divided by the number of parallel processors until the quotient remains greater than m (namely, until it becomes $O(m \ln m)$). In other words, we demonstrate that KARM(γ) and CG(γ) can be implemented on p processors with the time complexities, respectively,

$$\mathfrak{C}_p(\mathrm{KARM}(\gamma)) \leqq O(p^{-1}\mathfrak{C}_1(\mathrm{KARM}(\gamma)) + m \ln m),$$
$$\mathfrak{C}_p(\mathrm{CG}(\gamma)) \leqq O(p^{-1}\mathfrak{C}_1(\mathrm{CG}(\gamma)) + m \ln m).$$

The critical value of p depends on γ and the acceleration scheme we are going to parallelize, but this value is always greater in order than m. This result does not depend on the type ("standard" or "fast") of the involved linear algebra routines. The aim of this paper is to describe the Karmarkar and the conjugate-gradient-based acceleration strategies with "fast" linear algebra routines and to prove the above-mentioned result on the efficient parallelization of the corresponding algorithms.

The order of exposition is as follows. An outline of the ideas underlying acceleration schemes for the barrier method as applied to problem (1) is the subject of § 2. Section 3 contains necessary preliminary statements, in particular, the Karmarkar acceleration scheme. In § 4, the preconditioned conjugate-gradient-based acceleration originating from [NN 1989] is described.

2. Single-processor acceleration: Overview. As far as the single-processor computations are concerned, the only possibility for acceleration of the above procedure is to use, at a given iteration, the results of the auxiliary computations (like the inverse Hessians used to perform Newton steps) made on previous iterations. This is possible, since the matrices involved in the equations defining the Newton displacements at different iterations are not independent, but are closely related to each other. Therefore,

it is possible to reduce the average time complexity of an iteration using the previous inverse Hessians in order to compute the current one. This idea originates from Karmarkar [Ka 1984] and is implemented as follows. All the Hessians involved in Newton steps are of the form

$$M(d) = Z + A \operatorname{Diag}\{d\} A^T, \tag{23}$$

where Z is a fixed positive semidefinite $n \times n$ matrix, A is a fixed $n \times m$ matrix, and d is an m-dimensional vector with positive elements depending on the iteration. In order to update x in BASIC we substitute into (23)

$$d = d^*(t^+, x),$$

which leads to the precise Hessian of the function $F_{t^+}^{\phi}(x)$ and then invert this Hessian to obtain Q^+. It turns out that we could use approximate inverse Hessian instead of the precise one, provided that the approximation is compatible with the precise inverse Hessian within a factor of order one. Namely, it can be shown that the choice

$$d^+ := d^*(t^+, x) \tag{24}$$

in BASIC can be replaced by any other choice of d^+ satisfying the relation

$$\rho^{-1} \leqq d^+ / d^*(t^+, x) \leqq \rho \tag{25}$$

(where ρ is an appropriate *absolute constant*) without destroying any essential property of the procedure.

The idea underlying the Karmarkar acceleration is: to choose d^+ as the vector satisfying (25) with the greatest possible number of entries coinciding with the corresponding entries of the similar vector d, used at the previous iteration. Let k denote the number of entries of d^+ different from the corresponding entries of d (this number will be called the *rank* of the iteration). Then the approximate Hessian $M(d^+)$, which is to be inverted at the current iteration, is a k-rank correction of the similar matrix $M(d)$ inverted at the previous iteration; therefore, the matrix $Q^+ = (M(d^+))^{-1}$ can be represented as a k-rank correction of the matrix Q computed at the previous iteration, which allows us to find Q^+, for the case of k much less than m, much faster than by the direct inversion. To initialize this process, at the first iteration we, as in the procedure BASIC, set $d^+ = d^*(\tau^+, x_{\text{in}})$ and compute $M(d^+)$ and $Q^+ = (M(d^+))^{-1}$ directly (in other words, the rank of the initial iteration is, by definition, m).

The theoretical analysis of the time complexity of the above procedure is based on the fact that the sum of ranks $k(1), \cdots, k(N)$ of all the iterations in the procedure satisfies the relation

$$\sum_{s=1}^{N} k(s) = O(m) \tag{26}$$

while the number of iterations $N = O(m^{1/2})$. The latter relation immediately follows from the updating rule for t, and (26) is a consequence of the inequality

$$\| d^*(t^+, x^{\text{cr}}) / d^*(t, x^{\text{pr}}) - e \|_2 \leqq O(1), \tag{27}$$

where x^{pr} denotes the initial value of x at the previous iteration, x^{cr} is the initial value of x at the current iteration, t, t^+ are the corresponding values of the barrier parameter, and e is the m-dimensional vector of ones. Inequality (27), in turn, follows from (17).

It is not difficult to show that the single-processor time complexity of the sth iteration in the above procedure (where the "fast" linear algebra is implemented) is $O(m^2 k^{\gamma}(s))$ (this is the complexity of the updating $Q \mapsto Q^+$); all remaining operations

can be performed at a lower cost, namely, $O(m^2)$. Therefore, the resulting procedure KARM (γ) satisfies the relation

$$\mathfrak{C}_1(\text{KARM}(\gamma)) \leqq O(1) \max\left\{ \sum_{s=1}^{N} m^2(k^\gamma(s)+1) \,\middle|\, \sum_{s=1}^{N} k(s) = O(m) \right\} \tag{28}$$
$$= O(1)(m^{2+\gamma}N^{1-\gamma} + m^2 N) = O(m^{2.5+0.5\gamma})$$

(the latter equality holds since $N = O(m^{1/2})$).

How can one improve this result? Note that the computations performed at an iteration of KARM (γ) are, in a sense, unbalanced: at the sth iteration it requires $O(m^2 k^\gamma(s))$ units of time to update the approximate inverse Hessian, that is, $O(m^{2+0.5\gamma})$ units at average, and only $O(m^2)$ units to perform the remaining computations. In view of this observation, it seems reasonable to relax the requirements on the compatibility of the approximate and the precise inverse Hessians and to use an iterative procedure to compute approximate Newton direction. More precisely, assume that, when updating x, we first compute a symmetric matrix Q^+ such that

$$\rho(m)^{-1} Q(d^*(t^+, x)) \leqq Q^+ \leqq \rho(m) Q(d^*(t^+, x)) \tag{29}$$

for certain "large" $\rho(m)$ instead of $\rho(m) = O(1)$, as in the Karmarkar acceleration scheme. To update x, we need to find the Newton direction, that is, to solve the linear system

$$M(d^*(t^+, x))y = \nabla_x F^\phi_{t^+}(x) \tag{30}$$

with the unknown y. This system can be solved with the aid of the preconditioned conjugate-gradient method associated with the Euclidean structure on $\mathbb{R}^n$ defined by the scalar product $x^T(Q^+)^{-1}x$. This is equivalent to the solution of a linear system with an $n \times n$ symmetric positive definite matrix with the condition number $\text{Cond} = GO(\rho^2(m))$ using the standard conjugate-gradient method. It is well known that the rate of convergence of the latter method is

$$\exp\{-O(\text{Cond}^{-1/2})k\} = \exp\{-O(\rho^{-1}(m))k\},$$

k being the number of steps. It turns out that to maintain (17), we must ensure an appropriate absolute constant accuracy in solving (30). Therefore, we can restrict ourselves to $O(p(m))$ steps of the CG, which requires $O(p(m)m^2)$ units of time.

Now, the average time complexity of an iteration becomes

$$\{\text{average complexity of the updating } Q \mapsto Q^+ \text{ which ensures (29)}\} \tag{31}$$
$$+\{O(p(m)m^2)\}.$$

One can hope that the first term in the latter sum is lesser the greater $\rho(m)$ is, since increasing $\rho(m)$ means that we relax the requirements concerning the "distance" between the approximate and the precise inverse Hessians. At the same time, in the case of $\rho(m) = O(1)$, this term is equal to the average complexity $O(m^{2+0.5\gamma})$ of an iteration in KARM (γ). Thus, when increasing $\rho(m)$, one can reasonably decrease the total complexity of the updating of the approximate inverse Hessians in the procedure. Of course, we cannot take too large $\rho(m)$, due to the second term in (31). Nevertheless one can hope that the choice of $\rho(m)$ balancing the terms in (31) reduces in order the complexity of the procedure. That is the main idea underlying procedure CG (γ).

3. Procedure KARM (γ) and its parallelization.

3.1. Preliminary results on parallel "fast" linear algebra. Recall that we have fixed $\gamma \in (0, 1]$ such that for all $k \in \mathcal{N}$ the single-processor time complexity of multiplication

of a pair of $k \times k$ matrices does not exceed $O(k^{2+\gamma})$. It is then known (see [So 1988]) that

$$\mathfrak{C}_1(\text{Inversion of a nonsingular } k \times k \text{ matrix}) = O(k^{2+\gamma}); \tag{32}$$

$$\mathfrak{C}_p(\text{Multiplication of a pair of } k \times k \text{ matrices}) = O(k^{2+\gamma}/p + \ln k); \tag{33}$$

$$\mathfrak{C}_p(\text{Inversion of a nonsingular } k \times k \text{ matrix}) = O(k^{2+\gamma}/p + k \ln k) \tag{34}$$

(as above, $\mathfrak{C}_p(\Pi)$ denotes the p-complexity, that is, the p-processor time complexity of procedure Π).

The following lemmas are simple consequences of (32)–(34).

LEMMA 1. *Let $\sigma(l, k, r) = l\ k\ r\ (\min\{l, k, r\})^{\gamma-1}$ for $l, k, r \in \mathbb{N}$. The product of an $l \times k$ matrix A and a $k \times r$ matrix B can be computed with the p-complexity $O(\sigma(l, k, r)/p + \ln^2(\max\{l, k, r\}))$.*

Proof. Let $s = \min\{l, k, r\}$, $m = \max\{l, k, r\}$. Without loss of generality, we can assume that $s > 1$ and that l, k, r are divisible by s. Let $l' = l/s$, $k' = k/s$, $r' = r/s$, so that

$$\sigma(l, k, r) = (l'k'r')s^{2+\gamma}.$$

It suffices to prove that if $p = p_1 p_2$, where

$$p_2 =]s^{2+\gamma}/\ln m[, \qquad p_1 =]l'k'r'/\ln m[,$$

then the p-complexity of the computation of AB does not exceed $\ln^2 m$.

After partitioning the matrices A and B into square $s \times s$ submatrices, we obtain $l' \times k'$ and $k' \times r'$ matrices A', B' with elements from the ring $\mathfrak{L}$ of real $s \times s$ matrices. Let us regard our $p_1 p_2$ processors as p_1 macroprocessors, each macroprocessor comprising p_2 processors. The sum and the product of a pair of elements of $\mathfrak{L}$ can be computed with the p_2-complexity not exceeding $\tau = O(\ln m)$ (see (33)), or, in other words, can be computed at the same time by using a single macroprocessor. Since the standard multiplication of A' and B' with the aid of p_1 macroprocessors admits the optimal parallelization, the computation of A and B requires no more than $O(l'k'r'/p_1 + \ln m) \times \tau$ time units, so that the $p_1 p_2$-complexity of our computation does not exceed $O(\ln^2 m)$. □

From now on, $\mathfrak{S} \equiv \{d \in R^m \mid d > 0\}$.

The following lemma holds.

LEMMA 2. (i) *Given $x \in G'$, $t > 0$, and $d \in \mathfrak{S}$, we can* (a) *compute $\nabla_x F_t^{\phi}(x)$ with the p-complexity $O(mn/p + \ln m)$*; (b) *compute $d^*(t, x)$ with the p-complexity $O(mn/p + \ln m)$*; (c) *compute the product of $M(d)$ and a given vector $h \in R^n$ with the p-complexity $O(mn/p + \ln m)$*; (d) *compute $M(d)$ with the p-complexity $O(mn^{1+\gamma}/p + \ln^2 m)$*; (e) *compute $M(d)^{-1}$ with the p-complexity $O(mn^{1+\gamma}/p + n \ln n)$.*

(ii) *Assume that we have computed d, $d' \in \mathfrak{S}$ and the matrix $Q = [M(d)]^{-1}$, and let k be the number of positions in which the entries of d and d' do not coincide. Then $[M(d')]^{-1}$ can be computed with the p-complexity $O(m/p + n \max(n, k) \min^{\gamma}(n, k)/p + \ln^2 m + k \ln k)$.*

Proof. (i) The first and the second statements are evident; the third follows from the relation

$$M(d)h = \phi''h + (A(\mathrm{Diag}\{d\}(A^T h))).$$

The fourth statement follows from Lemma 1, since the p-complexity of the computation of the $m \times n$ matrix $\mathrm{Diag}\{d\}A^T$ is $O(mn/p + 1)$, the p-complexity of the multiplication of A and this matrix is $O(mn^{1+\gamma}/p + \ln^2 m)$, and it takes $O(n^2/p + 1)$ units of time to add ϕ'' to the result with the aid of p processors.

The fifth statement can be proved as follows. As we have seen, the p-complexity of computing $M(d)$ is $O(mn^{1+\gamma}/p+\ln^2 m)$. In accordance with (34) the resulting $n\times n$ matrix can be inverted with the p-complexity $O(n^{2+\gamma}/p+n\ln n)$.

(ii) Assume first that $p>1$. If $k=0$, then the statement is evident. Let k be a positive integer. It is clear that

$$M'_{nn}\equiv M(d')=M(d)+V_{nk}S_{kn}\equiv M_{nn}+V_{nk}S_{kn},$$

where V_{nk} and S_{kn} can be computed with the p-complexity $O(m/p+nk/p+\ln m)$.

Let $k\leqq n$. By the well-known formula, we have ($Q_{nn}=M(d)^{-1}$, subscripts denote the numbers of rows and columns)

$$[M'_{nn}]^{-1}=Q_{nn}-Q_{nn}V_{nk}(I_k+S_{kn}Q_{nn}V_{nk})^{-1}S_{kn}Q_{nn},$$

I_l being the $l\times l$ identity matrix. By Lemma 1, the matrix $(I_k+S_{kn}Q_{nn}V_{nk})$ can be computed with the p-complexity $O(n^2k^\gamma/p+\ln^2 m)+O(nk^{1+\gamma}/p+\ln^2 m)$; the resulting matrix can be inverted in $O(k^{2+\gamma}/p+k\ln k)$ units of time (see (34)). All the remaining estimates are based on Lemma 1. After $(I_k+S_{kn}Q_{nn}V_{nk})^{-1}$ is computed, it takes no more than $O(n^2k^\gamma/p+\ln^2 n)$ units of time to compute the matrix $Q_{nn}V_{nk}(I_k+S_{kn}Q_{nn}V_{nk})^{-1}$ and the same time to compute the matrix $S_{kn}Q_{nn}$. It takes no more than $O(nk^{1+\gamma}/p+\ln^2 n)$ units of time to compute the product of the latter two matrices and no more than $O(n^2/p+1)$ units to add the result to ϕ''. Thus (ii) holds in the case of $k\leqq n$.

Now let $k>n$. We have

$$(M'_{nn})^{-1}=(I_n+Q_{nn}V_{nk}S_{kn})^{-1}Q_{nn}.$$

The matrix $I_n+Q_{nn}V_{nk}S_{kn}$ can be computed with the p-complexity $O(kn^{1+\gamma}/p+\ln^2 m)$ (Lemma 1); the resulting matrix can be inverted with the p-complexity $O(n^{2+\gamma}/p+n\ln n)$ (see (34)), and, as above, it takes no more than $O(n^{2+\gamma}/p+\ln^2 n)+O(n^2/p)$ units of time to perform the remaining computations with the aid of p processors. □

3.2. The main inequality. To proceed, we are to formulate an important analytical property of the trajectory $x^\phi(\cdot)$. As far as we know, this property was first established in [NN 1989, Lem. 5.3 and Cor. 5.1].

LEMMA 3. *Let $t', t''\in[\tau,2\tau]$. Assume that $x', x''\in G'$ are such that*

$$\delta_t(x),\ \delta_{t'}(x')\leqq 2\mu\leqq 0.01,$$

and let

$$d'=d^*(t',x'),\qquad d''=d^*(t'',x'').$$

Then

$$\begin{aligned}&\{t't''\}^{1/2}(x'-x'')^T\phi''(x'-x'')+\|(\sqrt{d'}-\sqrt{d''})^2/\sqrt{d'd''}\|_1\\&\quad\leqq\mu_0^2\{m\{\sqrt{t'}-\sqrt{t''}\}^2/\sqrt{t't''}+1\}\end{aligned}\tag{35}$$

with an absolute constant μ_0.

We also use the following statement (the proof can be found in many papers on the Karmarkar acceleration of the path-following interior point methods).

LEMMA 4. *For an appropriate choice of positive absolute constants $\mu\leqq 0.05$, $\kappa>0$, and $\rho>1$, the following implication holds. Let $t>0$, $x\in G'$ satisfy the relation*

$$\delta_t(x)\leqq\mu;$$

let t^+ be such that

$$(1+\kappa\mu^{1/2}m^{-1/2})^{-1}\leqq t^+/t\leqq(1+\kappa\mu^{1/2}m^{-1/2}),$$

and let d^+ *be a positive vector such that*

$$\rho^{-1} \leqq d_i^+ / d_i^*(t^+, x) \leqq \rho.$$

Then the point

$$x^+ = x - (t^+)^{-1} M(d^+)^{-1} \nabla_x F_{t^+}^{\phi}(x)$$

belongs to G' *and the following relations hold*:

$$\delta_{t^+}(x) \leqq 2\mu; \qquad \delta_{t^+}(x^+) \leqq \mu.$$

3.3. Procedure KARM (γ). Now we can describe the p-processor implementation of KARM (γ).

INPUT: $x_{\text{in}} \in G'$ *such that*

$$\delta_\tau(x_{\text{in}}) \leqq \mu$$

(μ, κ, ρ *are defined in Lemma* 4).

INITIALIZATION:

(1) *Set*

$$x = x_{\text{in}};$$

$$t = \tau;$$

(2) *Compute*

$$d = d^*(\tau, x);$$

$$M(d) = \phi'' + A \operatorname{Diag}\{d\} A^T;$$

$$Q = M(d)^{-1}.$$

Note that, by virtue of Lemma 2,

(36) $$\mathfrak{C}_p(\text{INITIALIZATION}) = O(mn^{1+\gamma}/p + n \ln n + \ln^2 m).$$

ITERATION # s (input:
positive real t;
m-dimensional feasible x;
m-dimensional positive d;
$n \times n$ *symmetric positive definite* $Q = M(d)^{-1}$)

(1) *Set*

$$t^+ = \min\{(1 + \kappa m^{-1/2})t, 2\tau\};$$

(2) *Compute*

$$d^* = d(t^+, x)$$

and

$$g^+ = \nabla_x F_{t^+}^{\phi}(x);$$

(3) *Compute* d^+:

$$d_i^+ = \begin{cases} d_i, & \rho^{-1} \leqq d_i / d_i^* \leqq \rho, \\ d_i^*(t^+, x), & \textit{otherwise}; \end{cases}$$

(4) *Using* $Q = M(d)^{-1}$, *compute*

$$Q^+ = M(d^+)^{-1};$$

(5) *Update* x:

$$x^+ = x - (t^+)^{-1} Q^+ g^+;$$

(6) **IF** $t^+ = 2\tau$ **THEN** *set* $x_{\text{out}} = x$ *and terminate*
ELSE *set*

$$x = x^+$$

$$t = t^+$$

$$d = d^+$$

$$Q = Q^+$$

and go to the next iteration.

Note that in view of Lemma 2, the p-complexity of the sth iteration ($s = 1, 2, \cdots,$) satisfies the estimate

$$\begin{aligned} \mathfrak{C}_p(\text{ITERATION}) = O(mn/p + n \max(n, k(s)) \min^{\gamma}(n, k(s)), k(s))/p \\ + k(s) \ln k(s) + \ln^2 m, \end{aligned} \tag{37}$$

where $k(s)$ is the rank of the iteration, i.e.,

$$k(s) = |\{i \mid d_i(s) \neq d_i(s-1)\}|.$$

Herein $d(s)$ denotes the value attained by the vector d after the sth iteration, and $d(0) = d^*(t, x_{in})$. The main result on KARM (γ) is as follows.

THEOREM 1. *Procedure* KARM (γ) *maintains implication* (17) *and solves our Basic Problem. The p-complexity of the procedure satisfies the inequality*

$$\mathfrak{C}_p(\text{KARM}(\gamma)) = O(mn^{1+\gamma}/p + m^{3/2}n/p + n^2 m^{(1+\gamma)/2}/p + m \ln m). \tag{38}$$

Thus, in the case of relatively small n, that is, when

$$n \leqq m^{1-0.5\gamma}, \tag{39}$$

one has

$$\mathfrak{C}_p(\text{KARM}(\gamma)) = O(m^{3/2}n/p + m \ln m), \tag{40}$$

and the parallelization is efficient (that is, the acceleration due to parallelization is proportional to the number of processors) when the number of processors satisfies the inequality

$$p \leqq O(m^{1/2}n/\ln m). \tag{41}$$

In the case of relatively large n, that is, when

$$n > m^{1-0.5\gamma}, \tag{42}$$

one has

$$\mathfrak{C}_p(\text{KARM}(\gamma)) = O(m^{(1+\gamma)/2}n^2/p = m \ln m), \tag{43}$$

and the parallelization is efficient when the number of processors satisfies the inequality

$$p \leqq O(n^2 m^{(\gamma-1)/2}/\ln m); \tag{44}$$

in particular, the parallelization is efficient when $p \leqq O(m^{(3-\gamma)/2})$.

Proof. The validity of KARM (γ) (the fact that the procedure maintains (17) and therefore solves the Basic Problem) is a well-known fact (it is an immediate corollary of Lemma 4).

Let us estimate the complexity of the procedure. The following statement holds.

LEMMA 5. *Let* $t(0)=\tau$ *and let* $t(1), \cdots, t(K) \in [\tau, 2\tau]$ *be such that* $t(s)/t(s+1) \geqq (1+\kappa m^{-1/2})$, $1 \leqq s < K$. *Also, let* $x(s)$, $0 \leqq s \leqq K$, *be such that*

$$\delta_{t(0)}(x(0)) \leqq \mu; \qquad \delta_{t(s+1)}(x(s)) \leqq 2\mu, \quad 0 \leqq s < K.$$

Let $d(0)=d^*(t(0), x(0))$; *let* $d(s+1)$ *be defined as the vector with coordinates*

$$d_i(s+1)=\begin{cases} d_i(s), & \rho^{-1} \leqq d_i(s)/d_i^*(t(s+1), x(s)) \leqq \rho, \\ d_i^*(t(s+1), x(s)), & \text{otherwise}; \end{cases}$$

and let

$$k(s)=|\{i \mid d_i(s) \neq d_i(s-1)\}|.$$

Then

$$\sum_{s=1}^{K} k(s) \leqq O(m). \tag{45}$$

Proof of the lemma. Let $r(s)$ be the m-dimensional vector with the coordinates

$$|\ln (d_i^*(t(s), x(s-1))/d_i^*(t(s-1), x(s-2)))|, \qquad s \geqq 1,$$

where, by definition, $x(-1)=x(0)$. The updating rule for $d(s)$ clearly implies that

$$\sum_{s=1}^{K} k(s) \leqq O\left(\sum_{s=1}^{K} \|r(s)\|_1\right). \tag{46}$$

At the same time, (35), as applied to $t'=t(s-1)$, $x'=x(s-2)$, $t''=t(s)$, $x''=x(s-1)$, implies

$$\begin{aligned} \|r(s)\|_2^2 &\leqq O(m(t^{1/2}(s)-t^{1/2}(s-1))^2(t(s)t(s-1))^{-1/2}+1) \\ &\leqq O(m(t(s)-t(s-1))^2\tau^{-2}+1), \qquad s \geqq 1. \end{aligned}$$

Thus,

$$\|r(s)\|_1 \leqq O(m(t(s)-t(s-1))\tau^{-1}+m^{1/2}). \tag{47}$$

The required inequality (45) is an immediate corollary of (46), (47), and the assumption

$$t(s)/t(s-1) \geqq 1+O(m^{-1/2}), \qquad 0 \leqq s < K. \qquad \square$$

Note that, due to Lemma 4, the sequential values of t and x produced at iterations of KARM (γ) satisfy the premise in Lemma 5, the corresponding K being equal to $N=O(m^{1/2})$. Thus, the total rank $\sum_{s=1}^{N} k(s)$ of iterations of KARM (γ) does not exceed $O(m)$:

$$\sum_{s=1}^{N} k(s) \leqq O(m). \tag{48}$$

From (36), (37), it follows that

$$\begin{aligned} \mathfrak{C}_p(\text{KARM }(\gamma)) \leqq\ & \mathfrak{C}_p(\text{INITIALIZATION}) + O(m^{1/2})\{mn/p+\ln^2 m\} \\ & + \sum_{\{s \mid 1 \leqq s \leqq N,\, k(s) \leqq n\}} n^2 k^{\gamma}(s)/p + \sum_{\{s \mid 1 \leqq s \leqq N,\, k(s) > n\}} k(s) n^{1+\gamma}/p \\ & + \sum_{s=1}^{N} k(s) \ln k(s). \end{aligned}$$

In view of (48) and the relation $N=O(m^{1/2})$, the latter inequality implies (38). $\square$

4. Procedure CG(γ) and its parallelization. The idea underlying the conjugate-gradient-based acceleration was briefly explained in § 2. Here we present the detailed description of the procedure. It is reasonable to use this procedure in the case when n is not too small, namely, when

$$n \geqq m^{(1+\gamma-\gamma^2)/(1+\gamma)}, \tag{49}$$

otherwise, the procedure has no advantages in comparison to KARM (γ). Thus, from now on, we assume that (49) is satisfied.

In the description of CG (γ) below, κ and Λ denote appropriate positive *absolute* constants; the choice of these constants is discussed in [NN 1989, § 6].

Let

□
$$\rho = 10\{n^{(1+\gamma)} m^{-(1+\gamma-\gamma^2)}\}^{2/(2+3\gamma-\gamma^2)},$$

(50)
$$M =]\{m^{\gamma/2} n\}^{\gamma/(2+3\gamma-\gamma^2)}[,$$

$$K =]\ln^{-1}(1+\kappa m^{-1/2}) M^{-1}[, \qquad L = KM.$$

Let us start with some definitions. Let h be an m-dimensional vector with positive entries h_l, $1 \leqq l \leqq m$. For $1 \leqq l \leqq m$, let

$$\Gamma_j(h, l) = \{s > 0 \,|\, \rho^{2j-1} h_l \leqq s < \rho^{2j+1} h_l\}, \qquad j \in \mathbb{Z}.$$

The numbers $h_l \rho^{2j}$ will be called the *centers* of the *zones* $\Gamma_j(h, l)$. For a positive vector $d \in R^m$, the vector $\pi_h(d)$ is defined as a vector from R^m with the lth component, $1 \leqq l \leqq m$, being the center of the zone of the family $\{\Gamma_j(h, l) \,|\, j \in \mathbb{Z}\}$ that contains the number d_l.

The procedure CG (γ) is as follows (the quantities in angle brackets that accompany the rules are the p-complexities of the corresponding computations; these quantities are estimated with the aid of Lemma 2).

INITIALIZATION:

(1) *Set*

$$t = \tau;$$

$$x = x_{\text{in}};$$

(2) *Compute*

$$d = d^*(\tau, x_{\text{in}});$$

$$M = M(d);$$

$$Q = M(d)^{-1};$$

(3) *Set*

$$h = d; \qquad \langle O(mn^{1+\gamma}/p + n \ln n + \ln^2 m)\rangle$$

In what follows, it is convenient to split the iterations into sequential groups with M iterations in each group. It will be proved that the total amount of iterations in the procedure does not exceed L, so that the total amount of groups is not greater than K.

ITERATION # s (input: *positive real* t; *n-dimensionable feasible* x; *m-dimensional positive* d; *m-dimensional positive* h; $n \times n$ *symmetric positive definite* $Q = M(d)^{-1}$).

I. *Updating* $(t, h, d, Q) \mapsto (t^+, h^+, d^+, Q^+)$:

I.1. *Set*

$$t^+ = \min\{(1+\kappa m^{-1/2})t.2\tau\}; \qquad \langle O(1)\rangle$$

I.2. *Compute*

$$d^*(t^+, x); \qquad \langle O(mn/p + \ln m)\rangle$$

$$d' = \pi_h(d^*(t^+, x)); \qquad \langle O(m/p+1)\rangle$$

I.3. *Using Q, d, compute*

$$Q' = M(d')^{-1}$$

$$\langle O(m/p + n \max\{q(s), n\} \min^{\gamma}\{q(s), n\}/p + q(s) \ln q(s) + \ln^2 m),$$

where

$$q(s) = |\{i \,|\, 1 \leqq i \leqq m,\ d_i' \neq d_i\}|\rangle$$

I.4. **IF** *the iteration is not an initial iteration of a group*
THEN
set

$$h^+ = h;$$

$$d^+ = d';$$

$$Q^+ = Q';$$

ELSE
set for $i = 1, \cdots, m$:

$$d_i^+ = \begin{cases} d_i^*(t^+, x), & |\ln(d_i^*(t^+, x)/h_i)| > 1 \\ d_i', & \textit{otherwise}; \end{cases}$$

$$h_i^+ = \begin{cases} d_i^*(t^+, x), & |\ln(d_i^*(t^+, x)/h_i)| > 1 \\ h_i, & \textit{otherwise}; \end{cases}$$

Using Q' *to compute*

$$Q^+ = M(d^+)^{-1}$$

$$\langle O(m/p + n \max\{k(l), n\} \min^{\gamma}\{k(l), n\}/p + k(l) \ln q(l) + \ln^2 m),$$

if the iteration starts the lth group, where

$$k(l) = |\{i \,|\, 1 \leqq i \leqq m,\ d_i^+ \neq d_l'\}|;$$

otherwise the complexity is $O(n^2/p+1)\rangle$

II. *Updating* $x \mapsto x^+$:

II.1. *Perform*

$$N(\rho) =]\rho\Lambda[+ 1$$

Steps of the process

□

$$u_0 = 0;\ s_1 = r_0 = -Q^+ \nabla_x F_{t^+}^{\phi}(x);$$

$$r_j = r_{j-1} + \alpha_j Q^+ \nabla_x^2 F_{t^+}^{\phi}(x) s_j; \tag{51}$$

$$s_{j+1} = r_j + \beta_j s_j;$$

$$u_j = u_{j-1} + \alpha_j s_j,$$

where

□ *$u_\cdot$, $r_\cdot$, $s_\cdot$ are vectors from R^n;*

$$\alpha_j = -\frac{r_{j-1}^T M^+ r_{j-1}}{r_{j-1}^T \nabla_x^2 F_{t^+}^{\phi}(x) s_j};$$

$$B_j = \frac{r_j^T M^+ r_j}{r_{j-1}^T M^+ r_{j-1}};$$

$$M^+ = M(d^+); \qquad \langle N(\rho) O(mn/p + \ln m)\rangle$$

II.2. *Set*

$$x^+ = x - u_{N(\rho)}; \qquad \langle O(n/p+1)\rangle$$

III. *Renewal*:
IF

$$t^+ = \tau$$

THEN *set* $x_{\text{out}} = x^+$ *and terminate*
ELSE
set

$$t = t^+$$

$$d = d^+$$

$$h = h^+$$

$$Q = Q^+$$

and go to the next iteration.

Comment to II. Process (51) is the conjugate-gradient method (corresponding to the metric induced by the matrix M^+), as applied to the equation

$$\nabla_x^2 F_{t^+}^{\phi}(x) y = \nabla_x F_{t^+}^{\phi}(x).$$

Note that the computations are interrupted after $N(\rho)$ steps.

THEOREM 2. *The procedure* CG(γ) *maintains* (17) *for an appropriately chosen absolute constant* $\mu \leqq 0.05$ *and solves the Basic Problem. The p-complexity of the procedure under assumption* (49) *satisfies the relation*

$$\begin{aligned} \mathfrak{C}_p(\mathrm{CG}(\gamma)) &= O(m^{r_1(\gamma)} n^{r_2(\gamma)}/p + mn^{1+\gamma}/p + m \ln m) \\ &\leqq O(m^{r(\gamma)}/p + m \ln m), \end{aligned} \tag{52}$$

where

□

$$r_1(\gamma) = (2+5\gamma+\gamma^2)/(4+6\gamma-2\gamma^2),$$

$$r_2(\gamma) = (4+5\gamma-\gamma^2)/(2+3\gamma-\gamma^2),$$

$$r(\gamma) = r_1(\gamma) + r_2(\gamma) = 2.5 + 2\gamma^2/(2+3\gamma-\gamma^2).$$

In particular, in the case of $n = O(m)$, the parallelization of CG(γ) is efficient when the number of processors satisfies the relation

$$p \leqq O(m^{1.5+2\gamma^2/(2+3\gamma-\gamma^2)}/\ln m).$$

Proof. The validity of CG (γ) is proved in [NN 1989, § 6]; it is also shown that under the assumption (49) the following statements hold:

(i) The total number of iterations does not exceed

$$L = O(m^{1/2}); \tag{53}$$

(ii) The total number of groups of iterations does not exceed

$$K = O(m^{\sigma_1} n^{-\sigma_2}), \qquad \sigma_1 = \frac{2+3\gamma-2\gamma^2}{2(2+3\gamma-\gamma^2)}, \qquad \sigma_2 = \frac{\gamma}{2+3\gamma-\gamma^2}; \tag{54}$$

(iii) For each s one has

$$q(s) \leqq q^* = m^{\sigma_3} n^{-\sigma_4}, \qquad \sigma_3 = \frac{1+\gamma}{2+3\gamma-\gamma^2}, \qquad \sigma_4 = \frac{1-\gamma}{2+3\gamma-\gamma^2}, \tag{55}$$

so that

$$q^* \leqq n, \qquad q^* \leqq m^{1/2}; \tag{56}$$

(iv) The following relations hold:

$$\sum_{l=1}^{K} k(l) \leqq O(m); \tag{57}$$

$$\rho = O(m^{-\sigma_5} n^{2\sigma_3}), \qquad \sigma_5 = 2\frac{1+\gamma-\gamma^2}{2+3\gamma-\gamma^2}. \tag{58}$$

Now we can estimate the complexity of CG (γ).

From the "complexity comments" to the rules comprising CG (γ) and in view of (56), it is clear that

$$\begin{aligned} &\mathfrak{C}_p(C(\gamma)) \\ &\leqq O\Big(mn^{1+\gamma}/p + n \ln n + \ln^2 m \\ &\quad + m^{1/2}\{1 + mn/p + n^2(q^*)^\gamma/p + q^* \ln q^* + \rho mn/p + \rho \ln m + \ln^2 m\} \\ &\quad + \sum_{\substack{1 \leqq l \leqq K \\ k(l) \leqq n}} n^2 k^\gamma(l)/p + \sum_{\substack{1 \leqq l \leqq K \\ k(l) > n}} n^{1+\gamma} k(l)/p + \sum_{l=1}^{K} k(l) \ln k(l) + K \ln^2 m \Big). \end{aligned} \tag{59}$$

In view of (53)–(58), the right-hand side of (59) does not exceed

$$O(S/p + T),$$

where

$$\begin{aligned} S &= mn^{1+\gamma} + m^{3/2} n + n^{2-\gamma\sigma_4} m^{0.5+\gamma\sigma_3} + m^{1.5-\sigma_5} n^{1+2\sigma_3} + m^{\sigma_1(1-\gamma)+\gamma} n^{2-\sigma_2(1-\gamma)}, \\ T &= m \ln m + m^{0.5-\sigma_5} n^{2\sigma_3} \ln m + m^{\sigma_1} n^{\sigma_2} \ln^2 m. \end{aligned} \tag{60}$$

A straightforward computation shows that

$$S \leqq O(mn^{1+\gamma} + m^{r_1(\gamma)} n^{r_2(\gamma)}), \qquad T \leqq O(m \ln m),$$

which immediately leads to (52). □

REFERENCES

[GL1988] D. GOLDFARB AND S. LIU (1988), *An $O(n^3L)$ primal interior point algorithm for convex quadratic programming*, Tech. Report, Department of Industrial Engineering and Operations Research, Columbia University, New York, NY.

[Go1987] C. C. GONZAGA (1987), *An algorithm for solving linear programming problems in $O(n^3L)$ operations*, Tech. Report, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA; also appeared in *Progress in Mathematical Programming: Interior-Point and Related Methods*, N. Megiddo, ed., Springer-Verlag, Berlin, New York, 1989, pp. 1–28.

[Ka1984] N. KARMARKAR (1984), *A new polynomial-time algorithm for linear programming*, Combinatorica, 4, pp. 373–395.

[KMY1988] M. S. KOJIMA, S. MIZUNO, AND A. YOSHISE (1988), *An $O(n^{1/2}L)$ iteration potential reduction algorithm for linear complementarity problems*, Res. Reports on Information Sciences B-217, Department of Information Sciences, Tokyo Institute of Technology, Meguro-ku, Tokyo, Japan.

[KMY1989] ——— (1989), *A polynomial time algorithm for linear complementarity problems*, Math. Programming, 44, pp. 1–26.

[MA1989a] R. D. C. MONTEIRO AND I. ADLER (1989), *Interior path-following primal–dual algorithms. Part* I: *Linear programming*, Math. Programming, 44, pp. 27–42.

[MA1989b] ——— (1989), *Interior path-following primal–dual algorithms. Part* II: *Convex quadratic programming*, Math. Programming, 44, pp. 43–66.

[Ne1988a] Yu. E. NESTEROV (1988), *Polynomial-time methods in linear and quadratic programming*, Izv. Akad. Nauk SSSR Teckn. Kibernet., 3, pp. 3–6. (In Russian.)

[Ne1988b] ——— (1988), *Polynomial-time iterative methods in linear and quadratic programming*, Voprosy Kibernet. Moscow. (In Russian.)

[NN1989] Yu. E. NESTEROV AND A. S. NEMIROVSKY (1989), *Self-concordant functions and polynomial time methods in convex programming*, Central Economical and Mathematical Institute, U.S.S.R. Academy of Science, Moscow.

[Re1988] J. RENEGAR (1988), *A polynomial-time algorithm, based on Newton's method, for linear programming*, Math. Programming, 40, pp. 59–93.

[RV1989] C. ROOS AND J.-P. VIAL (1989), *Long steps with the logarithmic penalty barrier function in linear programming*, Report 89–44, Department of Mathematics and Computer Science, Delft University of Technology, Delft, the Netherlands.

[So1988] V. I. SOLODOVNIKOV (1988), *Parallel direct linear algebra algorithms*, Kibernet. i Vychisl. Teck., 4, pp. 32–86.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 565-582, November 1991

009

ORDERINGS FOR CONJUGATE GRADIENT PRECONDITIONINGS*

JAMES M. ORTEGA†

Abstract. Many preconditioners (e.g., SSOR, ILU) for the conjugate gradient method require the solution of sparse triangular systems of equations. For elliptic boundary value problems, one approach to obtaining additional parallelism in the solution of these systems is the use of red/black or multicolor orderings. There has been increasing evidence, however, that these orderings degrade the rate of convergence compared with the natural ordering. An alternative is the diagonal ordering, which maintains the rate of convergence of the natural ordering but has less parallelism than multicolor orderings. This paper reviews these as well as other orderings and then gives some results that help to explain why the red/black ordering gives an inferior rate of convergence.

Key words. preconditioned conjugate gradient, orderings, rate of convergence, red/black, SSOR, incomplete Cholesky, parallel and vector computing

AMS(MOS) subject classifications. 65F10, 65N20

1. Introduction.

We are concerned in this paper with the effect of orderings on the rate of convergence of the conjugate gradient method with SSOR or incomplete Cholesky (IC) preconditioning. We consider the linear system

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

which we assume arises from the discretization of a Poisson-type equation of the form

$$\nabla(\mathbf{K} \cdot \nabla u) = f \tag{1.2}$$

in two or three dimensions. Here $\mathbf{K}$ is a given vector-valued function of the spatial variables such that (1.2) is elliptic. For simplicity, we will restrict ourselves to rectangular or parallelepiped domains and Dirichlet boundary conditions, although many of the considerations are more general. We also assume that finite difference discretizations are done in such a way that A is symmetric positive definite, and has the usual five-diagonal structure in two dimensions and seven-diagonal structure in three dimensions.

The preconditioning step in the conjugate gradient method requires solving a subsidiary system of equations

$$M\tilde{\mathbf{r}} = \mathbf{r} \tag{1.3}$$

to obtain a modified residual vector $\tilde{\mathbf{r}}$. M is the preconditioning matrix, assumed to be symmetric positive definite, and for many commonly used preconditioners, such as SSOR or IC factorization, has the form

$$M = LDL^T, \tag{1.4}$$

where L is lower triangular and D is diagonal. Thus, the solution of (1.3) requires the solution of triangular systems with coefficient matrices L and L^T. In many cases, for example, SSOR and no-fill IC factorization, L will have the same nonzero structure as the lower triangular portion of A itself. The problem, then, is how to solve such triangular systems effectively on parallel and vector architectures.

* Received by the editors August 24, 1990; accepted for publication (in revised form) December 6, 1990. This research was supported in part by National Aeronautics and Space Administration grant NAG-1-1112-FDP.

† Department of Applied Mathematics, University of Virginia, Charlottesville, Virginia 22903.

There have been two main approaches to this problem. The first is to reorder the unknowns so that the coefficient matrix takes the form $\hat{A} = PAP^T$ for some permutation matrix P. The classical reordering is the red/black ordering (Young [1971]), in which

$$\hat{A} = \begin{bmatrix} D_1 & C^T \\ C & D_2 \end{bmatrix}, \tag{1.5}$$

where D_1 and D_2 are diagonal. The use of the red/black ordering for carrying out the SOR iteration on parallel and vector machines dates back to the early 1970's (Erickson [1972], Lambiotte [1975]). More recently, various "multicolor orderings" (see, e.g., Ortega [1988] for a review) have been used for both Poisson-type equations and more general equations with more general finite element or finite difference discretizations. For a multicolor ordering, the coefficient matrix takes the form

$$\hat{A} = \begin{bmatrix} D_1 & C_{21}^T & \cdots & C_{c1}^T \\ C_{21} & D_2 & & \\ \vdots & & \ddots & C_{c,c-1}^T \\ C_{c1} & \cdots & C_{c,c-1} & D_c \end{bmatrix}, \tag{1.6}$$

where, again, the D_i are diagonal, and c is the number of colors. For a multicolor ordering, the solution of a lower triangular system $L\mathbf{x} = \mathbf{d}$ of the same structure can be computed by

$$\mathbf{x}_i = D_i^{-1}\left(\mathbf{d}_i - \sum_{j<i} C_{ij}\mathbf{x}_j\right), \qquad i = 1, \cdots, c. \tag{1.7}$$

Since the D_i are diagonal, the solution has been reduced to matrix-vector multiplications, which are potentially ideal for parallel and vector machines. In particular, for the red/black ordering, (1.7) reduces to

$$\mathbf{x}_2 = D_2^{-1}(\mathbf{d}_2 - C\mathbf{x}_1). \tag{1.8}$$

For regular problems, C consists of only a few nonzero diagonals and the multiplication $C\mathbf{x}_1$ is efficiently executed on vector machines by multiplication by diagonals (Madsen, Rodrigue, and Karush [1976]).

Multicolor orderings exhibit a high degree of parallelism but may have a deleterious effect on the rate of convergence of iterative methods. For SOR itself, the asymptotic rate of convergence for the red/black ordering is the same as the natural ordering (Young [1971]) and the same result extends to many multicolor orderings (Adams and Jordan [1985]). Moreover, the use of multicolor orderings seems to enhance the rate of convergence in practice. Unfortunately, there has been growing evidence that the rate of convergence of the conjugate gradient method may be degraded, sometimes seriously, when such orderings are used for preconditioners (Poole and Ortega [1987], Ashcraft and Grimes [1988]). For example for (1.2), on a $63 \times 63 \times 63$ grid (250,000 unknowns), Harrar and Ortega [1990] reported 162 iterations for SSOR preconditioned conjugate gradient using the red/black ordering and 38 iterations using the natural ordering. Moreover, the red/black ordering is the basis for the reduced system conjugate gradient method (see, e.g., Hageman and Young [1981]), and since this method is mathematically equivalent to SSOR preconditioning on the original system (see, e.g., Harrar and Ortega [1990]), it suffers from the same rate-of-convergence problem.

A number of other reorderings have been considered but mostly with similar results. Duff and Meurant [1989] reported on an extensive set of experiments using incomplete Cholesky preconditioning for problem (1.2) on a 30×30 grid in two

dimensions, with problems containing anisotropy, discontinuous coefficients, etc. They considered 16 different reordering strategies, of which only six gave rates of convergence comparable to the natural ordering on all problems, and three of these orderings are equivalent to the natural ordering.

The second main approach to solving the triangular systems effectively is to obtain what parallelism is available in the natural ordering. The basic idea is exemplified by the diagonal ordering shown in Fig. 1.1 in two dimensions.

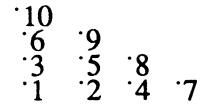


FIG. 1.1. *Diagonal ordering of grid points.*

In carrying out SOR with a five-point stencil, the unknowns on each diagonal can be updated in parallel or with vector operations whose lengths are the number of points in a diagonal. This is true because the five-point stencil couples unknowns only from different diagonals. However, the updates produced from this ordering are exactly the same as with the natural ordering. For example, the update at point 5 in Fig. 1.1 depends only on the updated values at 2 and 3 and the old values at 8 and 9; thus, it makes no difference whether all the unknowns on the first row are updated before 5 is or not. This basic idea extends to three dimensions and is also related to multicolor orderings, as will be discussed in the next section. In two dimensions, this diagonal ordering was studied in Young [1971] as an example of a consistent ordering and discussed as a possible paradigm for vectorization by Hayes [1978]. It has been explored in both two and three dimensions in increasingly sophisticated ways by van der Vorst [1983], [1989a], [1989b], Schlichting and van der Vorst [1989], and Ashcraft and Grimes [1988]. We will discuss the diagonal ordering in more detail in the next section, including extensions to more general discretizations.

Given the experimental results that certain orderings may degrade the rate of convergence, why is this the case? One attempt at an explanation was Melhem [1986] and the concept of "zero-stretch." He observed that orderings, such as the red/black ordering, that moved elements of the matrix away from the main diagonal tended to degrade the rate of convergence. However, we follow here the lead of Duff and Meurant [1989], who computed the "remainder matrices" for their different orderings. We discuss in § 4, for a few particular orderings, the structure and size of these remainder matrices. Prior to that, we collect in § 3 some results on estimation of the condition number as a function of reorderings.

Other results relevant to the general problem of the effect of reorderings on the rate of convergence are given in D'Azevedo, Forsyth, and Tang [1990]; Doi and Lichnewsky [1990]; and Eijkhout [1990].

2. Diagonal and related orderings. In this section, we expand on the discussion of diagonal orderings, as exemplified by Fig. 1.1. As observed by Poole and Ortega [1987], the fact that points on a diagonal are not coupled with themselves implies that the ordering of Fig. 1.1 is a multicolor ordering if we assign a separate color to each diagonal. The coefficient matrix in this ordering has the form shown in Fig. 2.1, which is taken from Stotland [1990]. Note that this ordering is just the Cuthill-McKee (CM) [1969] ordering for bandwidth/profile minimization. Duff and Meurant [1989] also

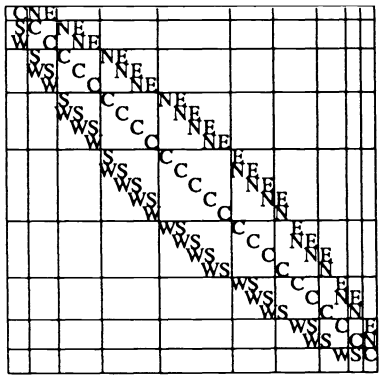


FIG. 2.1. *Matrix for diagonal ordering (two dimensions).*

consider the reverse CM ordering, in which the numbering proceeds from the opposite vertex of the grid, as well as a block CM ordering. These are also equivalent to the natural ordering.

Figure 2.1 illustrates the vectors that can be used to carry out the multiplications of (1.7). In particular, the vector lengths are $1, \cdots, N-1, N, N-1, \cdots, 1$ on an $N \times N$ grid. These are considerably smaller than the corresponding vector lengths in the red/black ordering, which are $O(N^2/2)$. But Ashcraft and Grimes [1988] have given results on a CRAY X-MP indicating that the diagonal ordering is superior to the red/black ordering on that machine. However, Elman and Agron [1989] and Chan, Kuo, and Tong [1989] have shown that red/black or multicoloring orderings will probably be superior on highly parallel machines. In particular, the first of these two papers gives theoretical results for a hypercube architecture, and the second gives theoretical and computational results for a Connection Machine. Thus, the situation seems to be that the diagonal ordering will be superior on machines requiring only a modest degree of parallelism or vectorization, but as the parallelism of the architecture increases, red/black or multicoloring orderings may become relatively more competitive.

We next discuss a problem with the diagonal ordering and a way to alleviate it that was observed by van der Vorst [1989a]. For the reordered matrix of Fig. 2.1, we no longer have the long diagonals of the naturally ordered matrix to use in the matrix-vector multiply in the conjugate gradient method. In particular, there are breaks in the diagonals, corresponding to the different C_{ij} of (1.6). On the other hand, we can leave the matrix in the naturally ordered form and still carry out the updates according to the diagonal ordering paradigm. This is the approach taken by Ashcraft and Grimes [1988], but the vectors now have stride $N-1$, which may cause memory bank conflicts on Cray machines. Thus, in the natural ordering, the data is arranged optimally for the matrix multiply but not for the preconditioning and vice-versa for the diagonal ordering. However, as observed by van der Vorst [1989a], we can circumvent this problem by means of the Eisenstat modification [1981], which eliminates the need for the matrix multiply. Thus, we can use the diagonal ordering of Fig. 2.1 so as to have suitable vectors for the preconditioning.

Ordering by diagonals is a special case of *wavefront* methods, which can be applied to more general triangular systems; see Greenbaum [1986] and Saltz [1990]. Rather than the general case, we next discuss another particular discretization. Consider the nine-point stencil shown in Fig. 2.2, which is used for fourth-order approximations to

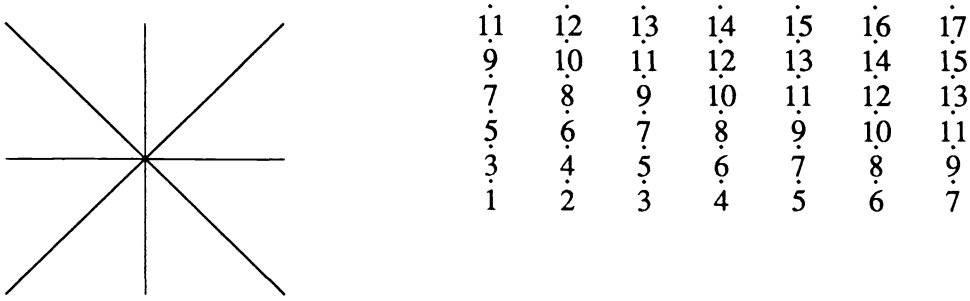


FIG. 2.2. *Nine-point stencil and multicolor "diagonal" ordering.*

Poisson's equation or for equations containing a mixed derivative u_{xy}. A multicolor "diagonal" ordering for the nine-point stencil is also shown in Fig. 2.2.

Again for SOR, unknowns corresponding to grid points of the same number in Fig. 2.2 can be updated simultaneously. For example, after unknowns 1 and 2 have been updated, both of the unknowns labeled 3 can be updated simultaneously, since the only updated unknowns on which they depend are 1 and 2. The only difference between this situation and the previous one with the five-point stencil is that now the set of grid points of color i is no longer a diagonal of adjacent grid points but, rather, a bent "diagonal" of points separated by a "knight's move," as shown in Fig. 2.2.

If the unknowns are ordered corresponding to the multicoloring of Fig. 2.2, the coefficient matrix is again a multicolored matrix and takes the form

$$A = \begin{bmatrix} D_1 & L_{12} & L_{13} & L_{14} & & & \\ & \ddots & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ & \text{Symm} & & \ddots & \ddots & \ddots & \\ & & & & \ddots & \ddots & \\ & & & & & \ddots & \\ & & & & & & D_c \end{bmatrix}. \tag{2.1}$$

The D_i are diagonal: D_1 and D_2 are 1×1 matrices, D_3 and D_4 are 2×2, and so on to the maximum size, which is $O(N/2)$, after which they decrease in size. More precisely, on an $N\times N$ grid, if N is even, there are $3N-2$ colors (i.e., $c=3N-2$ in (1.6)) and the maximum size of the D_i is $N/2$, which is taken on by $N+2$ of the D_i. If N is odd, $c=3N$ and the maximum size of the D_i is $(N+1)/2$, which is taken on by $(N+3)/2$ of the D_i.

The off-diagonal matrices $L_{i,i+2}$ and $L_{i,i+3}$ in (2.1) each have a single nonzero diagonal whose length is the same as the corresponding D_i. Except for the 1×1 matrices L_{12} and $L_{c-1,c}$, the matrices $L_{i,i+1}$ each have two nonzero diagonals and the precise structure depends on whether i is odd or even, as shown in (2.2) and (2.3):

$$L_{ii+1} = \quad i \text{ odd}, \qquad \frac{i+1}{2}\times\frac{i+1}{2} \tag{2.2}$$

$$L_{ii+1} = \quad i \text{ even}, \qquad \frac{i}{2}\times\frac{i}{2}+1. \tag{2.3}$$

Thus $L_{i,i+1}$ has diagonals of length $i/2$ if i is even, and $(i+1)/2$ and $(i-1)/2$ if i is odd; this holds until the $L_{i,i+1}$ reach their maximum size and then start decreasing. Note that $L_{i,i+1}$ is square if i is odd but rectangular if i is even, as shown by the matrix sizes given in (2.2) and (2.3). This is illustrated in Fig. 2.3, which shows the first few blocks of a typical matrix.

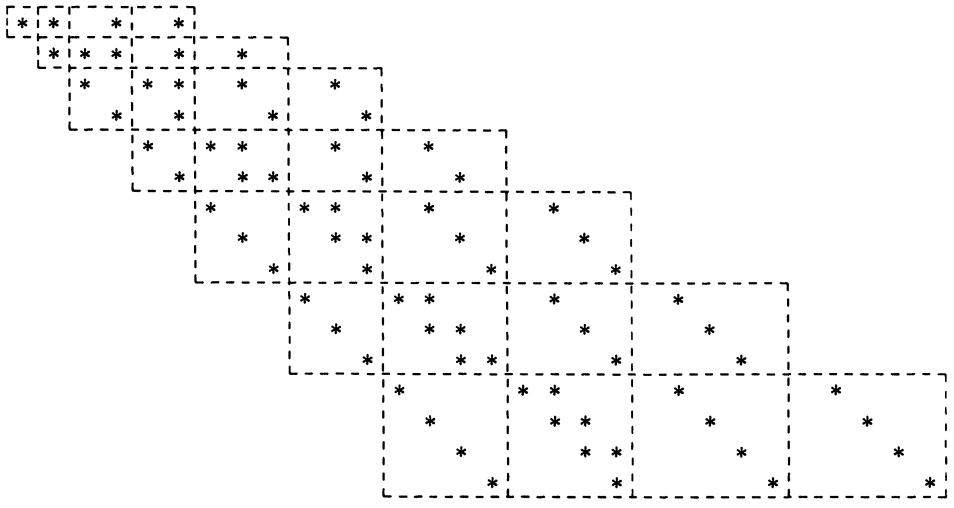


FIG. 2.3. *Structure of matrix.*

We next comment on the differences in vector lengths between the five-point and nine-point stencils. With the nine-point stencil we have noted that the maximum size of a D_i is $O(N/2)$; this is the maximum vector length if we do not attempt to couple diagonals between adjacent blocks. This contrasts to the maximum vector length of $O(N)$ for the five-point stencil. Thus, the use of the nine-point stencil approximately halves the vector lengths, as compared with the five-point stencil. This is analogous to the use of the red/black ordering for the five-point stencil, which gives vector lengths of $O(N^2/2)$, compared with a four-color ordering for the nine-point stencil, which gives vector lengths of $O(N^2/4)$—nominally half the vector length of the red/black ordering.

We next consider three-dimensional problems. For simplicity we will restrict ourselves to Poisson-type equations (1.1) and the seven-point stencil. This stencil is the natural extension to three dimensions of the five-point stencil for two dimensions.

Ordering the grid points by diagonals in two dimensions extends naturally to ordering by diagonal planes for three dimensions. (See Ashcraft and Grimes [1988] and van der Vorst [1989a].) This is illustrated in Fig. 2.4 for a $3\times3\times3$ grid. Only two diagonal planes are shown in Fig. 2.4 but there are seven such planes, including two that consist of only a single point. The grid points in these seven planes are shown in Fig. 2.5. The first three planes in Fig. 2.5 correspond to the planes shown in Fig. 2.4, including the first plane, which contains only one point.

Points in a plane are assigned the same color. For the $3\times3\times3$ example, there are 7 planes and, hence, 7 colors. For an $N\times N\times N$ grid there are $3N-2$ planes and $3N-2$ colors. If we order the grid points, and therefore the unknowns, corresponding to these diagonal planes, we again obtain a multicolor block tridiagonal matrix. The

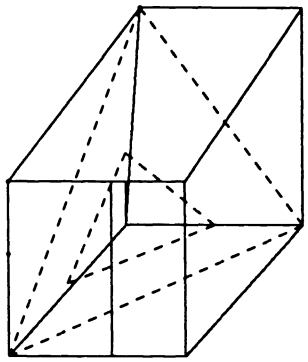


FIG. 2.4. *Ordering by diagonal planes*: $3\times3\times3$ *grid.*

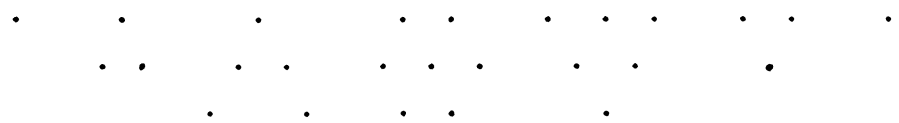


FIG. 2.5. *Grid points in diagonal planes*: $3\times3\times3$ *grid.*

detailed structure of the matrix in the case of a $4\times4\times4$ grid is shown in Fig. 2.6, which is taken from Stotland [1990].

The maximum number of grid points in a plane is $3N^2/4$ if N is even and $(3N^2+1)/4$ if N is odd. Since all points in a plane can be updated by SOR in parallel, the maximum degree of parallelism is $O(3N^2/4)$. However, an examination of Fig. 2.6 shows that the same is not true of the vector lengths. Consider one of the off-diagonal blocks of maximum size. Although there is one long diagonal, the other diagonals break up into lengths that are associated with two-dimensional problems. There is

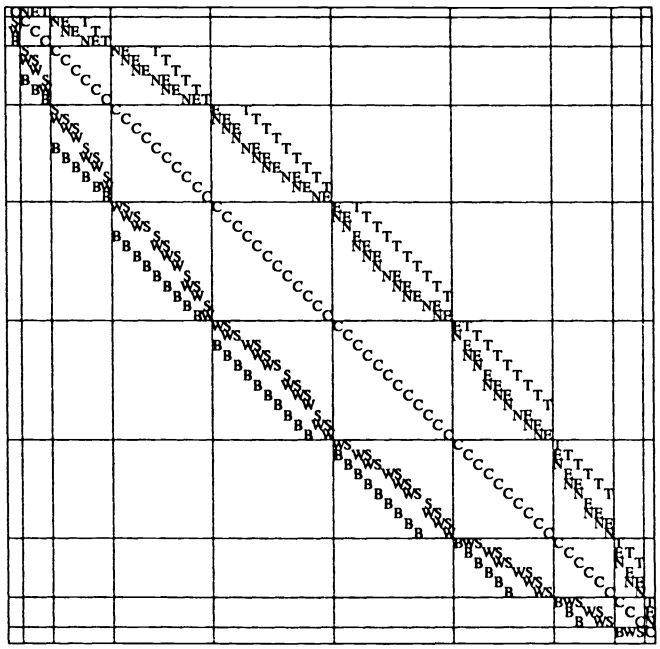


FIG. 2.6. *Matrix for diagonal ordering*: *Three dimensions,* 4×4 *grid.*

some overlapping of the diagonals across the submatrices corresponding to the two-dimensional problems so that some vectors of length greater than N can be obtained. Moreover, the gather operation can be used on some of the shorter vectors. However, in three dimensions the vectorization properties of the diagonal ordering are not as good as the parallel properties. This is in contrast to the two-dimensional situation.

We next consider orderings based on domain decomposition. Fig. 2.7(a) shows a standard four-domain ordering in which the points in each subdomain are ordered row-wise in such a way that within each domain the ordering proceeds from the outer corner inward. In Fig. 2.7(b), the subdomain points are ordered diagonally, so that vector or parallel properties of the diagonal ordering can be used within each subdomain. Duff and Meurant [1989] attribute these orderings within the subdomains to van der Vorst (see also van der Vorst [1987]). In either case, the points in the separator set are numbered last and with these orderings the coefficient matrix takes the familiar arrowhead form

$$A = \begin{bmatrix} A_1 & & & & B_1^T \\ & A_2 & & & B_2^T \\ & & A_3 & & B_3^T \\ & & & A_4 & B_4^T \\ B_1 & B_2 & B_3 & B_4 & A_S \end{bmatrix}. \tag{2.4}$$

Domain decomposition orderings have potentially good parallel properties, and were suggested by Farhat [1986] as a way to parallelize the SOR iteration. They can be used in the same way for SSOR or IC preconditioning. In particular, if there are p processors, it would be convenient to have p subdomains. Consider, for example, SSOR. In the solution of the corresponding lower triangular systems, each processor could solve one of the systems

$$L_i \mathbf{x}_i = \mathbf{d}_i \tag{2.5}$$

and form its contribution $B_i \mathbf{x}_i$ to the final system

$$L_S \mathbf{x}_S = \mathbf{b}_S - \sum_{i=1}^{p} B_i \mathbf{x}_i. \tag{2.6}$$

Note that the additions in the right-hand side of (2.6) require a fan-in, and the parallel solution of (2.6) is somewhat of a bottleneck.

In conjunction with SSOR or IC preconditioning, it is not necessary to have the separator set. If the separator points in Fig. 2.7 are incorporated into the four subdomains and the subdomains are numbered counterclockwise, the coefficient matrix

19	20	21	46	30	29	28
22	23	24	47	33	32	31
25	26	27	48	36	35	34
37	38	39	49	42	41	40
7	8	9	45	18	17	16
4	5	6	44	15	14	13
1	2	3	43	12	11	10

(a)

19	20	22	39	31	29	28
21	23	25	43	34	32	30
24	26	27	47	36	35	33
40	44	48	49	46	42	38
6	8	9	45	18	17	15
3	5	7	41	16	14	12
1	2	4	37	13	11	10

(b)

FIG. 2.7. *Domain decomposition orderings.*

now takes the form

$$A = \begin{bmatrix} A_1 & B_1^T & 0 & B_4^T \\ B_1 & A_2 & B_2^T & 0 \\ 0 & B_2 & A_3 & B_3^T \\ B_4 & 0 & B_3 & A_4 \end{bmatrix}, \tag{2.7}$$

where the A_i and B_i are not the same as in (2.4). For such nonseparator domain decomposition orderings, Duff and Meurant [1989] reported at least as good convergence results as for the natural ordering. (These orderings are called vdv1 and vdv2 in that paper, corresponding to row-wise or diagonal ordering within the subdomains.) Unfortunately, without the separator sets, the natural parallelism of (2.4) is lost. However, it is still possible to work separately within each subdomain. For example, on the forward sweep of SSOR, SOR may be applied separately in each subdomain. When values from an adjacent subdomain are needed, the old ones are used. This is no longer SSOR on the whole domain, however; it is block Jacobi with respect to the subdomains and SSOR within each subdomain. Thus, the splitting of A for the forward SOR sweep is

$$A = \begin{bmatrix} D_1 - L_1 & & & \\ & D_2 - L_2 & & \\ & & D_3 - L_3 & \\ & & & D_4 - L_4 \end{bmatrix} - \begin{bmatrix} U_1 & -B_1^T & 0 & -B_4^T \\ -B_1 & U_2 & -B_2^T & 0 \\ 0 & -B_2 & U_3 & -B_3^T \\ -B_4 & 0 & -B_3 & U_4 \end{bmatrix}, \tag{2.8}$$

where $D_i - L_i - U_i$ is the splitting of A_i into its diagonal and strictly triangular parts. Although this approach has perfect parallelism, the rate of convergence is likely to be inferior.

3. Condition numbers. In this section, we collect some basic results on the condition number of the preconditioned matrix. If M is the preconditioning matrix, we will work primarily with $M^{-1}A$, which is similar to the preconditioned matrix $M^{-1/2}AM^{-1/2}$ for the conjugate method. We define the remainder matrix R by

$$A = M - R. \tag{3.1}$$

Intuitively, the "smaller" R is, the better M approximates A and the smaller the condition number of the preconditioned matrix should be. More precisely,

$$M^{-1}A = I - M^{-1}R, \tag{3.2}$$

and if the spectral radius $\rho(M^{-1}R)$ is small, then the eigenvalues of $M^{-1}A$ will all be close to 1, so that $M^{-1}A$ will be well conditioned. Note that $M^{-1}R$ is the iteration matrix for a stationary linear iteration $\mathbf{x}^{k+1} = M^{-1}R\mathbf{x}^k + \mathbf{d}$ generated by the splitting (3.1), and rapid convergence of this iteration is equivalent to a better-conditioned matrix $M^{-1}A$. (For the conjugate gradient method, the distribution of the eigenvalues of $M^{-1}A$ is also important.) We next give a more precise relation between the condition number of $M^{-1}A$ and the eigenvalues of $M^{-1}R$.

If A and M are symmetric positive definite, then the eigenvalues of $M^{-1}A$ are positive, and

$$\text{cond}\,(M^{-1}A) = \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{3.3}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ are the maximum and minimum eigenvalues of $M^{-1}A$. (Note that $M^{-1}A$ is not necessarily symmetric, but since it is similar to the preconditioned

symmetric matrix $M^{-1/2}AM^{-1/2}$, it is customary to write cond $(M^{-1}A)$ as the l_2 condition number of $M^{-1/2}AM^{-1/2}$.) For the following theorem, recall that (3.1) is a regular splitting (Varga [1962]) if $M^{-1} \geqq 0$ and $R \geqq 0$, and a weak regular splitting (see, e.g., Ortega and Rheinboldt [1970]) if $M^{-1} \geqq 0$ and $M^{-1}R \geqq 0$.

THEOREM 3.1. *If A and M are symmetric positive definite and the spectral radius $\rho(M^{-1}R) < 1$, then*

$$\text{cond}\,(M^{-1}A) = \frac{1-\mu_{\min}}{1-\mu_{\max}} \tag{3.4}$$

where $\mu_{\max}$ and $\mu_{\min}$ are the maximum and minimum eigenvalues of $M^{-1}R$. If, in addition, A is an M-matrix and (3.1) *is a weak regular splitting, then*

$$\text{cond}\,(M^{-1}A) = [1+\rho(A^{-1}R)](1-\mu_{\min}). \tag{3.5}$$

Proof. By (3.2), the eigenvalues λ of $M^{-1}A$ and μ of $M^{-1}R$ are related by $\lambda = 1-\mu$. Since the eigenvalues of $M^{-1}A$ are all positive and since $\rho(M^{-1}R) < 1$, the eigenvalues of $M^{-1}R$ lie in the interval $[0, 1)$. Hence

$$\lambda_{\min} = 1-\mu_{\max}, \qquad \lambda_{\max} = 1-\mu_{\min},$$

and (3.4) follows. For the second part, by a theorem in Varga [1962] (proved for regular splittings but the same proof holds for weak regular splittings)

$$\rho(M^{-1}R) = \frac{\rho(A^{-1}R)}{1+\rho(A^{-1}R)}. \tag{3.6}$$

Thus,

$$1-\mu_{\max} = \frac{1}{1+\rho(A^{-1}R)}$$

and (3.5) follows. This completes the proof.

One can make the estimate

$$1-\mu_{\min} \leqq 1+|\mu_{\min}| \leqq 1+\rho(M^{-1}R) \tag{3.7}$$

and obtain the following corollary that was proved by Axelsson and Eijkhout [1989] under the weaker assumption that A and M are not necessarily symmetric, but that the eigenvalues of $M^{-1}R$ are real.

COROLLARY 3.1. *Under the assumptions of Theorem* 3.1,

$$\text{cond}\,(M^{-1}A) \leqq 1+2\rho(A^{-1}R). \tag{3.8}$$

The proof of this corollary follows from (3.7) and (3.5) by using (3.6) to obtain the identity

$$1+\rho(M^{-1}R) = \frac{1+2\rho(A^{-1}R)}{1+\rho(A^{-1}R)}.$$

Meijerink and van der Vorst [1977] showed that a no-fill ILU factorization of an M-matrix is a regular splitting. The same is true for the SSOR splitting with $\omega = 1$:

$$M = (D-L)D^{-1}(D-L^T), \qquad R = LD^{-1}L^T, \tag{3.9}$$

where L is the strictly lower triangular part of A, and D is the diagonal part. Thus, if A is symmetric positive definite, Theorem 3.1 and Corollary 3.1 apply to both of these preconditioners. Moreover, for (3.9), we can obtain a much sharper result.

COROLLARY 3.2. *If the conditions of Theorem* 3.1 *hold and R is positive semidefinite, then*

$$\text{cond}\,(M^{-1}A) \leqq 1+\rho(A^{-1}R); \tag{3.10}$$

and if R is singular, then

$$\text{cond}\,(M^{-1}A) = 1+\rho(A^{-1}R). \tag{3.11}$$

In particular, (3.11) *holds for* (3.9).

Proof. The eigenvalues of R, and hence $M^{-1}R$, are nonnegative so that $1-\mu_{\min} \leqq 1$; thus, (3.10) follows from (3.5). If R is singular, then $\mu_{\min}=0$ so that (3.11) is (3.5) in this case. Finally, since L in (3.9) is strictly lower triangular, R is singular.

We note that Corollary 3.2 does not apply to incomplete Cholesky factorization, even when $R \geqq 0$, since R need not be positive semidefinite. Moreover, the weaker but sufficient condition that the eigenvalues of $M^{-1}R$ be nonnegative does not necessarily hold either, as is seen by the simplest example of the Poisson equation and no-fill incomplete factorization.

4. Effect of ordering. We now wish to consider the effects of ordering on the rate of convergence. Let A_N be the coefficient matrix in the natural ordering and $A_C = PA_NP^T$, where P is some permutation matrix. If $A_N = M_N - R_N$ and $A_C = M_C - R_C$ are the corresponding splittings (3.1), then

$$A_C^{-1}R_C = (PA_NP^T)^{-1}R_C = PA_N^{-1}P^TR_CPP^T. \tag{4.1}$$

Thus, $A_C^{-1}R_C$ and $A_N^{-1}P^TR_CP$ are permutationally similar so that

$$\rho(A_C^{-1}R_C) = \rho(A_N^{-1}P^TR_CP), \qquad \|A_C^{-1}R_C\|_2 = \|A_N^{-1}P^TR_CP\|_2. \tag{4.2}$$

Therefore, the difference in the estimates (3.8)

$$\text{cond}\,(M_N^{-1}A_N) \leqq 1+2\rho(A_N^{-1}R_N) \leqq 1+2\|A_N^{-1}\|_2\|R_N\|_2 \tag{4.3}$$

$$\text{cond}\,(M_C^{-1}A_C) \leqq 1+2\rho(A_N^{-1}P^TR_CP) \leqq 1+2\|A_N^{-1}\|_2\|R_C\|_2 \tag{4.4}$$

depends only on the difference between $\|R_N\|_2$ and $\|R_C\|_2$ in the case of the norm estimates. However, these norm estimates may be overly pessimistic and the more critical factors are $\rho(A_N^{-1}R_N)$ and $\rho(A_N^{-1}P^TR_CP)$. This is especially the case when Corollary 3.2 applies, so that we have the exact condition numbers.

We first consider under what conditions

$$\rho(A_N^{-1}R_N) = \rho(A_N^{-1}P^TR_CP). \tag{4.5}$$

Clearly, a sufficient condition is

$$R_N = P^TR_CP, \tag{4.6}$$

which is equivalent to

$$M_N = P^TM_CP, \tag{4.7}$$

and we will use (4.7) and (4.6) interchangeably. We note that, in fact, the eigenvalues of $M_N^{-1}A_N$ and $M_C^{-1}A_C$ are the same if (4.7) holds for any nonsingular matrix P since

$$M_N^{-1}A_N = (P^TM_CP)^{-1}(P^TA_CP) = P^{-1}M_C^{-1}A_CP,$$

so that $M_N^{-1}A_N$ and $M_C^{-1}A_C$ are similar.

We next consider an example of when (4.6) holds. For the SSOR splitting (3.9), where $A_N = D_N - L_N - L_N^T$ and $A_C = D_C - L_C - L_C^T$, (4.6) may be written as

$$L_ND_N^{-1}L_N^T = P^TL_CPP^TD_C^{-1}PP^TL_C^TP. \tag{4.8}$$

Since $D_N = P^T D_C P$, (4.8) holds if

$$L_N = P^T L_C P. \tag{4.9}$$

For incomplete Cholesky factorization, we have a similar condition; in this case, if $\tilde{L}_N$ and $\tilde{L}_C$ are the factors, then

$$R_N = \tilde{L}_N \tilde{L}_N^T - A_N, \qquad R_C = \tilde{L}_C \tilde{L}_C^T - A_C,$$

so that (4.6) is

$$\tilde{L}_N \tilde{L}_N^T - A_N = P^T \tilde{L}_C P P^T \tilde{L}_C^T P - P^T A_C P.$$

Thus a sufficient condition for (4.6) to hold is

$$\tilde{L}_N = P^T \tilde{L}_C P. \tag{4.10}$$

For a no-fill incomplete factorization, $\tilde{L}_N$ will have the same nonzero off-diagonal structure as L_N, and similarly for $\tilde{L}_C$ and L_C. Thus (4.10) will hold if (4.9) does. Young [1971] has defined a permutation matrix P to be nonmigratory if (4.9) holds, and has shown that the permutation matrix that transforms the natural ordering to the diagonal ordering of Fig. 1.1 is nonmigratory. Hence, these orderings are equivalent (as is intuitively clear).

We can characterize the relation (4.9) in the following way, in terms of the Gauss-Seidel iterations

$$D_N \mathbf{u}_N^{k+1} - L_N \mathbf{u}_N^{k+1} - U_N \mathbf{u}_N^k = \mathbf{b}_N \tag{4.11a}$$

$$D_C \mathbf{u}_C^{k+1} - L_C \mathbf{u}_C^{k+1} - U_C \mathbf{u}_C^k = \mathbf{b}_C. \tag{4.11b}$$

We can rewrite (4.11b) as

$$P^T D_C P P^T \mathbf{u}_C^{k+1} - P^T L_C P P^T \mathbf{u}_C^{k+1} - P^T U_C P P^T \mathbf{u}_c^k = P^T \mathbf{b}_C. \tag{4.12}$$

Thus if (4.9) holds (and, consequently, $U_N = P^T U_C P$ also), then (4.12) becomes

$$D_N P^T \mathbf{u}_C^{k+1} - L_N P^T \mathbf{u}_C^{k+1} - U_N P^T \mathbf{u}_C^k = \mathbf{b}_N. \tag{4.13}$$

Therefore, provided that $\mathbf{u}_N^0 = P^T \mathbf{u}_C^0$, (4.11a) and (4.13) generate exactly the same sequences $\mathbf{u}_N^k$ and $P^T \mathbf{u}_C^k$, so that

$$\mathbf{u}_C^k = P \mathbf{u}_N^k. \tag{4.14}$$

On the other hand, if (4.9) does not hold, (4.11a) and (4.11b) will not generate sequences for which (4.14) holds, and the rates of convergence may be much different. Duff and Meurant [1989] tabulated the maximum element of R and the Frobenius norm of R $(=(\sum r_{ij}^2)^{1/2})$ for several different orderings on some two-dimensional Poisson-type model problems of the form (1.2) using ICCG. While these numbers are interesting, they are only qualitatively related to differences in the rates of convergence. The key quantity is $\rho(A^{-1}R)$, as discussed in the previous section. Figure 4.1 compares these quantities for SSOR ($\omega = 1$) preconditioning on the problem (1.2) with $K \equiv 1$ for 7×7

	$1+\rho(A^{-1}R)$	$\max \lvert r_{ij} \rvert$	$\lVert R \rVert_F$
Nat (7×7)	3.99	.125	.89
RB (7×7)	6.83	.25	2.56
Nat (15×15)	13.7	.125	4.7
RB (15×15)	26.3	.25	13.9

FIG. 4.1. SSOR *preconditioning.*

and 15×15 grids on the unit square. The coefficient matrix A has been scaled to have unit main diagonal. For this problem, (3.11) holds and thus $1+\rho(A^{-1}R)$ gives the exact condition number.

As seen in Fig. 4.1, $\max|r_{ij}|$ gives very little indication of the difference in the condition numbers. $\|R\|_F$ is better in that it increases as the condition number increases, but it is of marginal value in ascertaining the size of the condition number.

Even though R by itself does not determine the condition number accurately, it is certainly a key factor and we would like to understand how and why R changes for different orderings. In the sequel, we will assume that A arises from the five-point discretization of the Poisson-type problem (1.2) on an $N \times N$ grid, and that M has the form

$$M = (\bar{D} - L)\bar{D}^{-1}(\bar{D} - L^T), \tag{4.15}$$

where $\bar{D}$ is diagonal and L is the strictly lower triangular part of A. This will be the case for no-fill Cholesky factorization (van der Vorst [1982]) or symmetric Gauss–Seidel (SGS), but not for SSOR with $\omega \neq 1$. Then

$$R = M - A = \bar{D} - D + L\bar{D}^{-1}L^T, \tag{4.16}$$

where D is the main diagonal of A. Note that in the case of SGS, $\bar{D} = D$ so that $R = LD^{-1}L^T$.

We next obtain the structure of R by performing the multiplications $L\bar{D}^{-1}L^T$ in the following way. The ith row of $L\bar{D}^{-1}L^T$ is $\mathbf{e}_i^T L\bar{D}^{-1}L^T$, where $\mathbf{e}_i$ is the ith unit vector. In the natural ordering, the ith row of L has two nonzero elements and may be written as

$$\mathbf{e}_i^T L = a_{i,i-1}\mathbf{e}_{i-1}^T + a_{i,i-N}\mathbf{e}_{i-N}^T, \tag{4.17}$$

where the a_{ij} are off-diagonal elements of A. Then

$$\mathbf{e}_i^T L\bar{D}^{-1}L^T = a_{i,i-1}\bar{d}_{i-1}^{-1}\mathbf{e}_{i-1}^T L^T + a_{i,i-N}\bar{d}_{i-N}^{-1}\mathbf{e}_{i-N}^T L^T.$$

Now

$$\mathbf{e}_{i-1}^T L^T = a_{i,i-1}\mathbf{e}_i^T + a_{i+N-1,i-1}\mathbf{e}_{i+N-1}^T$$

and

$$\mathbf{e}_{i-N}^T L^T = a_{i-N+1,i-N}\mathbf{e}_{i-N+1}^T + a_{i,i-N}\mathbf{e}_i^T.$$

Thus

$$\begin{aligned}\mathbf{e}_i^T L\bar{D}^{-1}L^T = {} & a_{i,i-1}^2\bar{d}_{i-1}^{-1}\mathbf{e}_i^T + a_{i,i-1}a_{i+N-1,i-1}\bar{d}_{i-1}^{-1}\mathbf{e}_{i+N+1}^T \\ & + a_{i,i-N}a_{i-N+1,i-N}\bar{d}_{i-N}^{-1}\mathbf{e}_{i-N+1}^T + a_{i,i-N}^2\bar{d}_{i-N}^{-1}\mathbf{e}_i^T.\end{aligned} \tag{4.18}$$

The expression (4.18) for the ith row of $L\bar{D}^{-1}L^T$ shows that R is a three-diagonal matrix, in which the two off-diagonals are at a distance $N-1$ from the main diagonal; that is, they are shifted one position in, relative to the outermost diagonals of A itself. The magnitudes of the elements of R are given by (4.18) and (4.16):

$$\text{off-diagonals: } a_{i,i-1}a_{i+N-1,i-1}\bar{d}_{i-1}^{-1}, \quad a_{i,i-N}a_{i-N+1}\bar{d}_{i-N}^{-1} \tag{4.19a}$$

$$\text{diagonal: } a_{i,i-1}^2\bar{d}_{i-1}^{-1} + a_{i,i-N}^2\bar{d}_{i-N}^{-1} + \bar{d}_i - a_{ii}. \tag{4.19b}$$

Consider next the red/black ordering in which

$$A = \begin{bmatrix} D_1 & C^T \\ C & D_2 \end{bmatrix}, \qquad L = \begin{bmatrix} 0 & 0 \\ C & 0 \end{bmatrix},$$

so that

$$R = \bar{D} - D + \begin{bmatrix} 0 & 0 \\ 0 & C\bar{D}_1^{-1}C^T \end{bmatrix}.$$

Elman and Golub [1988] have computed the reduced system $CD_1^{-1}C^T$ for a somewhat different equation, but their general conclusions hold here also. In particular, $C\bar{D}_1^{-1}C^T$ is a nine-diagonal matrix corresponding to the stencil shown in Fig. 4.2. (Note that the points of the original five-point stencil, except for the center point, do not appear in the stencil for R. This would not be the case, however, for SSOR with $\omega \neq 1$.)

$$\begin{array}{ccccc} & & i+\dot{2}N & & \\ & i+\dot{N}-1 & \cdot & i+\dot{N}+1 & \\ i\dot{-}2 & \cdot & \dot{i} & \cdot & i\dot{+}2 \\ & i-\dot{N}-1 & \cdot & i-\dot{N}+1 & \\ & & i-\dot{2}N & & \end{array}$$

FIG. 4.2. *Nine-point stencil for R: Red/black ordering.*

The number of nonzero elements in $C\bar{D}_1^{-1}C^T$ is $0((9/2)N^2 - 8N)$. Thus R has either this number of nonzero elements or $0(N^2/2)$ more in case $D \neq \bar{D}$. In contrast, R for the natural ordering has $0(3N^2 - 2N)$ nonzero elements, so that the red/black remainder matrix has about 50 percent more nonzero elements, for large N. Moreover, the magnitudes of the nonzero elements in the red/black remainder matrix are larger, as we now discuss.

The elements of R for the natural ordering are given by (4.19). For the red/black ordering, we will not give the elements precisely, but will indicate their magnitude as follows. The elements of $\bar{D}^{-1}L^T$ are of the form $d^{-1}a$, where a is an element of A and d is an element of $\bar{D}$. When $\bar{D}^{-1}L^T$ is multiplied by $\mathbf{e}_i^T L$, the elements of $\bar{D}^{-1}L^T$ are multiplied by another element of A to give terms of the form $d^{-1}aa$. The important thing is that some of these terms may add together. In particular, the diagonal elements of $L\bar{D}^{-1}L^T$ will consist of a sum of four terms. Elements in the ith row corresponding to the four grid points $i \pm N \pm 1$ in Fig. 4.2 will consist of a sum of two terms, and the elements corresponding to the grid points $i \pm 2N$ and $i \pm 2$ consist of a single term. Comparing this with (4.19), we would expect that, roughly, the diagonal elements of R corresponding to $L\bar{D}^{-1}L^T$ for the red/black ordering will be twice as large as the diagonal elements of R for the natural ordering, and half of the off-diagonal elements will also be twice as large. If we assume that the elements of the R for the natural ordering all have the same magnitude r, then the Frobenius norm is $\|R\|_{F,Nat} \doteq \sqrt{3}\, Nr$, based on the estimate of $0(3N^2)$ elements in R. For the red/black ordering, we would have $0(N^2/2)$ elements with magnitude $4r$, $0(2N^2)$ with magnitude $2r$, and $0(2N^2)$ with magnitude r. Thus, $\|R\|_{F,RB} \doteq \sqrt{18}\, Nr$, so that the ratio of the Frobenius norms is about 2.5. This very rough estimate agrees quite well with the data in Fig. 4.1 for SSOR preconditioning, in which the ratios are about 3, and with three of the four

problems in Duff and Meurant [1989] for ICCG for which the ratios are about 2. (The fourth problem has very strong anisotropy and the ratio is 43.)

We comment on the difference between SOR and SSOR in terms of the remainder matrix. Consider the Gauss-Seidel splitting $A=(D-L)-L^T$. Here, in analogy to the splitting $A=M-R$, $D-L$ corresponds to M and L^T corresponds to R. In terms of the natural and red/black orderings, L_N^T and L_{RB}^T have the same number of elements and, in the case of Poisson's equation, exactly the same elements, just in different positions. Hence, the ordering has no effect on the "size" of R. For SSOR ($\omega=1$), however, the remainder matrices

$$(D_N-L_N)D_N^{-1}(D_N-L_N^T), \qquad (D_{RB}-L_{RB})D_{RB}^{-1}(D_{N_{RB}^{-1}}-L_{RB}^T)$$

are very different, as we have just seen, with the red/black remainder matrix having many more, as well as larger, elements.

We consider one more class of orderings: those based on domain decomposition, as discussed in § 2. Consider first the ordering of Fig. 2.7(a) with four domains and a separator set, so that the coefficient matrix A is given by (2.4). Then the strictly lower triangular part of A is

$$L=\begin{bmatrix} L_1 & & & & \\ & L_2 & & & \\ & & L_3 & & \\ & & & L_4 & \\ B_1 & B_2 & B_3 & B_4 & L_5 \end{bmatrix},$$

where L_i is the strictly lower triangular part of A_i. The diagonal blocks of $L\bar{D}^{-1}L^T$ are then

$$L_i\bar{D}_i^{-1}L_i^T, \qquad i=1,\cdots,4, \qquad L_5\bar{D}_5^{-1}L_5^T+\sum_{i=1}^{4} B_i\bar{D}_i^{-1}B_i^T \tag{4.20}$$

and the off-diagonal blocks are

$$L_i\bar{D}_i^{-1}B_i^T, \qquad i=1,\cdots,4, \qquad B_i\bar{D}_i^{-1}L_i^T, \qquad i=1,\cdots,4. \tag{4.21}$$

For simplicity, we will assume that each subdomain has $N_1\times N_1$ grid points so that $N=2N_1+1$, and there are $4N_1+1$ points in the separator set.

The matrices $L_i\bar{D}_i^{-1}L_i^T$, $i=1,\cdots,4$, have the three-diagonal structure for the natural order, as discussed earlier. The matrix L_5 has the form

$$L_5=\begin{bmatrix} L_{5,1} & & & & \\ & L_{5,2} & & & \\ & & L_{5,3} & & \\ & & & L_{5,4} & \\ \mathbf{v}_1^T & \mathbf{v}_2^T & \mathbf{v}_3^T & \mathbf{v}_4^T & 0 \end{bmatrix},$$

where each $L_{5,i}$ is lower bidiagonal (with zero main diagonal) and the $\mathbf{v}_i$ are column vectors with a nonzero element in the last position. It follows that $L_5\bar{D}_5^{-1}L_5^T$ is diagonal: each diagonal element is zero or of the form $d^{-1}aa$ except the last element, which is a sum of four such terms. As before, a denotes some element of A, and d some element of $\bar{D}$.

The matrix B_i^T gives the connections of the points of domain i with the separator points and the only nonzero rows correspond to interior grid points adjacent to the

separator points. Consider B_1; the situation for the other B_i is similar. The nonzero rows of B_1 corresponding to points along the vertical and horizontal separator sets are

$$(4.22)\quad a\mathbf{e}_i^T,\quad i=N_1,2N_1,\cdots,N_1^2,\qquad a\mathbf{e}_i^T,\quad i=N_1^2,N_1^2-1,\cdots,N_1^2-N_1+1,$$

where the element a of A is different in each case. Thus, B_1 may be expressed as a sum of rank one matrices of the form $a\mathbf{e}_j\mathbf{e}_i^T$, where i is one of the indices of (4.22) and j denotes the position of the corresponding row in B_1. Therefore, $B_1\bar{D}_1^{-1}B_1^T$ is a sum of terms of the form

$$aad^{-1}\mathbf{e}_{j_1}\mathbf{e}_{i_1}^T\mathbf{e}_{i_2}\mathbf{e}_{j_2}^T,$$

and these matrices are nonzero only when $i_1=i_2=i$, where i is one of the indices of (4.22). Hence, $B_1\bar{D}_1^{-1}B_1^T$ consists of $2N_1$ nonzero elements of the form aad^{-1}. Likewise, the other matrices $B_i\bar{D}_i^{-1}B_i^T$ have $2N_1$ nonzero elements.

Finally, consider the matrix $L_1\bar{D}_1^{-1}B_1^T$. The ith row of $L_1\bar{D}_1^{-1}$ is, in general, of the form

$$ad^{-1}\mathbf{e}_{i-1}^T+ad^{-1}\mathbf{e}_{i-N}^T.$$

Thus the ith row of $L_1\bar{D}_1^{-1}B_1^T$ is a linear combination of the $(i-1)$st and $(i-N_1)$th rows of B_1^T. But both of these rows cannot be simultaneously nonzero, so that $L_1\bar{D}_1^{-1}B_1^T$ has no more nonzero elements that B_1 itself. Similarly, the other $L_i\bar{D}_i^{-1}B_i^T$ have no more nonzero elements than B_i.

If we add the number of nonzero elements of each submatrix in $L\bar{D}^{-1}L^T$, we have at most

$$(4.23)\qquad 4\cdot 3N_1^2+(4N_1+4\cdot 2N_1)+8\cdot 2N_1=12N_1^2+28N_1$$

nonzero elements. The first term in (4.23) gives the number of nonzero elements in the first four matrices of (4.20), the second term is for the last matrix of (4.20), and the third term is for the eight matrices of (4.21). We have previously shown that the remainder matrix for the natural ordering has approximately $3N^2$ nonzero elements. Since $N=2N_1+1$, this gives $12N_1^2+12N_1+1$, which is only slightly smaller than (4.23) for large N. Thus, the remainder matrices for the natural and domain decomposition orderings have roughly the same number of nonzero elements. Although the above discussion has only been for the case of four subdomains, one would expect that the same conclusions would hold in more generality.

Suppose, next, that we remove the separator sets so that A has the form (2.7). In this case

$$L=\begin{bmatrix} L_1 & & & \\ B_1 & L_2 & & \\ 0 & B_2 & L_3 & \\ B_4 & 0 & B_3 & L_4 \end{bmatrix}$$

and the remainder matrix has the following submatrices:

$$(4.24)\qquad \begin{aligned} &L_1\bar{D}_1^{-1}L_1^T,\ L_i\bar{D}_i^{-1}L_i^T+B_{i-1}\bar{D}_{i-1}^{-1}B_{i-1}^T,\qquad i=2,3,\\ &L_4\bar{D}_4^{-1}L_4^T+B_4\bar{D}_4^{-1}B_4^T+B_3\bar{D}_3^{-1}B_3^T \end{aligned}$$

$$(4.25)\qquad L_i\bar{D}_i^{-1}B_i^T,\qquad i=1,2,3,\qquad L_1\bar{D}_1^{-1}B_4^T,\ B_1\bar{D}_1^{-1}B_4^T,$$

plus the transposes of the matrices in (4.25). The $L_i\bar{D}_i^{-1}L_i^T$ have the same natural order structure as before, and the B_i have the same general structure as in the separator case, since they provide the connections along the horizontal and vertical interfaces of the subdomains. A rough estimate of the nonzero elements in $L\bar{D}^{-1}L^T$ is then

$$3N_1^2+2(3N_1^2+2N_1)+3N_1^2+2\cdot 2N_1+10\cdot 2N_1=12N_1^2+28N_1. \tag{4.26}$$

Since now $N=2N_1$, $3N^2=12N_1^2$ is an estimate of the number of nonzero elements in the natural order remainder matrix. Again, for large N, this is very close to the estimate (4.26). This tends to explain why Duff and Meurant [1989] obtained essentially the same number of iterations for their domain decomposition orderings (the "*vdv*" orderings) as for the natural ordering.

Acknowledgment. I am indebted to my student Narinder Nayar for the computations of Fig. 4.1 and to two referees for some additional references and suggestions to improve the paper.

REFERENCES

L. ADAMS AND H. JORDAN [1985], *Is* SOR *color-blind?*, SIAM J. Sci. Statist. Comput., 7, pp. 490–506.

C. ASHCRAFT AND R. GRIMES [1988], *On vectorizing incomplete factorization and* SSOR *preconditioners*, SIAM J. Sci. Statist. Comput., 9, pp. 122–151.

O. AXELSSON AND V. EIJKHOUT [1989], *Vectorizable preconditioners for elliptic difference equations in three space dimensions*, J. Comput. Appl. Math., 27, pp. 299–321.

T. CHAN, C. C. KUO AND C. TONG [1989], *Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine*, Comput. Phys. Comm., 53, pp. 237–252.

E. CUTHILL AND J. MCKEE [1969], *Reducing the bandwidth of sparse symmetric matrices*, in Proc. 24th Nat. Conf. ACM, pp. 157–172.

E. D'AZEVEDO, P. FORSYTH, AND W.-P. TANG [1990], *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, Computer Science Report CS-90-04, Computer Science Department, University of Waterloo, Waterloo, Ontario, Canada.

S. DOI AND A. LICHNEWSKY [1990], *An ordering theory for incomplete LU factorizations on regular grids*, INRIA Report, Institut de Recherche d'Informatique et d'Automatique, Le Chesnay, France.

E. DUFF AND G. MEURANT [1989], *The effect of ordering on preconditioned conjugate gradients*, BIT, 29, pp. 635–657.

V. EIJKHOUT [1990], *Vectorizable and parallelizable preconditioners for the conjugate gradient method*, Ph.D. thesis, University of Nijmegen, the Netherlands.

S. EISENSTAT [1981], *Efficient implementation of a class of conjugate gradient methods*, SIAM J. Sci. Statist. Comput., 2, pp. 1–4.

H. ELMAN AND E. AGRON [1989], *Ordering techniques for the preconditioned conjugate gradient method on parallel computers*, Comput. Phys. Comm., 53, pp. 253–269.

H. ELMAN AND G. GOLUB [1988], *Iterative methods for cyclically reduced non-self-adjoint linear systems*, Computer Science Report CS-TR-2145, Computer Science Department, University of Maryland, College Park, MD.

J. ERICKSEN [1972], *Iterative and direct methods for solving Poisson's equation and their applicability to* ILLIAC IV, Tech. Report 60, Center for Advanced Computation, University of Illinois at Urbana-Champaign, Urbana, IL.

C. FARHAT [1986], *Multiprocessors in computational mechanics*, Ph.D. thesis, Civil Engineering Department, University of California, CA.

A. GREENBAUM [1986], *Solving sparse triangular linear systems using Fortran with parallel extensions on the NYU ultracomputer prototype*, Ultracomputer Note 99, New York University, New York.

L. HAGEMAN AND D. YOUNG [1981], *Applied Iterative Methods*, Academic Press, New York.

D. HARRAR AND J. ORTEGA [1990], *Solution of three-dimensional generalized Poisson equations on vector computers*, Iterative Methods for Large Linear Systems, D. Kincaid and L. Hayes, eds., Academic Press, New York, pp. 173–191.

L. HAYES [1978], *Timing analysis of standard iterative methods on a pipeline computer*, Report CNA-136, Center for Numerical Analysis, University of Texas, Austin, TX.

J. LAMBIOTTE [1975], *The solution of linear systems of equations on a vector computer*, Ph.D. thesis, Department of Applied Mathematics, University of Virginia, Charlottesville, VA.

N. MADSEN, G. RODRIGUE, AND J. KARUSH [1976], *Matrix multiplication by diagonals on a vector/parallel processor*, Inform. Process Lett., 5, pp. 41–45.

J. MEIJERINK AND H. VAN DER VORST [1977], *An iterative solution for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31, pp. 148–162.

R. MELHEM [1986], *Toward efficient implementation of preconditioned conjugate gradient methods on vector supercomputers*, Internat. J. Supercomputer Appl., 2, pp. 70–98.

J. ORTEGA [1988], *Introduction to parallel and vector solution of linear systems*, Plenum Press, New York.

J. ORTEGA AND W. RHEINBOLDT [1970], *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York.

E. POOLE AND J. ORTEGA [1987], *Multicolor* ICCG *methods for vector computers*, SIAM J. Numer. Anal., 24, pp. 1394–1418.

J. SALTZ [1990], *Aggregation methods for solving sparse triangular systems on multiprocessors*, SIAM J. Sci. Statist. Comput., 11, pp. 123–145.

J. SCHLICHTING AND H. VAN DER VORST [1989], *Solving 3D block bidiagonal linear sytems on vector computers*, J. Comput. Appl. Math., 27, pp. 323–330.

S. STOTLAND [1990], *Vectorizing the conjugate gradient method with preconditioning using the diagonal ordering*, Master's Project, Department of Applied Mathematics, University of Virginia, Charlottesville, VA.

H. VAN DER VORST [1982], *A vectorizable variant of some* ICCG *methods*, SIAM J. Sci. Statist. Comput., 3, pp. 350–356.

——— [1983], *On the vectorization of some simple* ICCG *methods*, First Internat. Conference on Vector and Parallel Computation in Scientific Applications, Paris.

——— [1987], *Large tridiagonal and block tridiagonal linear systems on vector and parallel computers*, Parallel Comput., 5, pp. 45–54.

——— [1989a], *High performance preconditioning*, SIAM J. Sci. Statist. Comput., 10, pp. 1174–1185.

——— [1989b], ICCG *and related methods for 3D problems on vector computers*, Comput. Phys. Comm., 53, pp. 223–235.

R. VARGA [1962], *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, NJ.

G. WITTUM [1989], *On the robustness of* ILU *smoothing*, SIAM J. Sci. Statist. Comput., 10, pp. 699–717.

D. YOUNG [1971], *Iterative solution of large linear systems*, Academic Press, New York.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 583–602, November 1991

010

AN INTERIOR POINT METHOD FOR BLOCK ANGULAR OPTIMIZATION*

GARY L. SCHULTZ† AND ROBERT R. MEYER†

Abstract. An interior point method for block angular optimization is developed and the convergence properties of the method are described. A major motivation for such a method is that most of the computation is easily parallelized. Computational results are presented for a class of large-scale linear programming models. These models are multicommodity flow problems that arise from an Air Force (Military Airlift Command) application and generate problems as large as 100,000 rows and 300,000 columns.

Key words. large-scale linear programming, block angular optimization, multicommodity network flow, parallel numerical methods, interior point methods

AMS(MOS) subject classifications. 90C06, 65Y05

1. Block angular optimization.

A block angular optimization problem is defined as the following: Given a smooth, convex function $c : \mathbb{R}^N \to \mathbb{R}$, find a minimizer x^* of c subject to

$$(1)\qquad \begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_K \\ D & & & \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{pmatrix} \begin{matrix} = \\ = \\ \vdots \\ = \\ \leq \end{matrix} \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \\ d \end{pmatrix}$$

and bounds $\ell \leq x \leq u$. The dimension of the kth block A_k is $M_k \times N_k$ and we define $M := \sum_k M_k$ and $N := \sum_k N_k$. The dimension of D is $J \times N$. The vectors x_k, b_k, etc. conform to these dimensions. Note that linear multicommodity problems are an important subclass of this class of problems.

We assume that solving a set of K linear subproblems:

$$(2)\qquad \left.\begin{array}{ll} \text{minimize} & \tilde{c}_k x_k \\ \text{subject to} & A_k x_k = \tilde{b}_k \\ & \tilde{\ell}_k \leq x_k \leq \tilde{u}_k \end{array}\right\} \quad \text{for } k = 1, \cdots, K$$

is easy relative to solving the entire original problem (1). This is quite reasonable for most applications, since the problems (2) are independent subproblems, and may, therefore, be solved in parallel. This is particularly appropriate in a MIMD computational environment. Second, some types of structure in the A_k may be exploited which could not be directly used if (1) were solved with the entire set of constraints. An example of this is the multicommodity flow problem, where each A_k is a node-arc-incidence matrix. In this case (2) may be solved with a special purpose network code. And third, we note that the difficulty of solving most linear programs (in some sense the easiest of optimization problems) in practice increases as a quadratic or

* Received by the editors September 26, 1990; accepted for publication (in revised form) March 15, 1991. This research was supported in part by National Science Foundation grant CCR-8907671 and Air Force Office of Scientific Research grant 89-0410.

† Computer Science Department, University of Wisconsin, 1210 West Dayton St., Madison, Wisconsin 53706.

cubic function with the size of the problem, so that it is much more efficient to solve a set of small problems than a single aggregate problem.

2. The decomposition scheme. In this section we describe a scheme that allows us to deal with the block constraints explicitly and the coupling constraints implicitly via a barrier function. Define the set of feasible points for the block constraints by

$$\mathcal{B} := \{x | Ax = b \text{ and } \ell \leq x \leq u\},$$

and the set of feasible points for the coupling constraints by

$$\mathcal{C} := \{x | Dx \leq d\}.$$

The algorithm begins by finding x^0 as the solution of a relaxed problem

$$\underset{x}{\text{minimize}}\ \tilde{c}x \quad \text{subject to } x \in \mathcal{B}. \tag{3}$$

Here $\tilde{c}$ could approximate the gradient of the original cost function $\nabla c(\tilde{x})$ at some point $\tilde{x}$, but we require only that $x^0 \in \mathcal{B} \cap \operatorname{dom} c$. The case where $\operatorname{dom} c \neq \mathbb{R}^N$ will not be considered further in this paper, as it produces technical difficulties without being enlightening. In subsequent iterations, the algorithm solves additional subproblems with block constraints, and then does a smaller (typically K-dimensional) search to coordinate the solutions of the subproblems, forming a new point in $\mathcal{B}$. During this process information about $\mathcal{C}$ is introduced into the objective function by using a barrier function.

We shall introduce the barrier function, discuss its fundamental properties, and show how to obtain feasible points in §2.1. Section 2.2 will show how to generate a sequence of feasible points that approximates the solution. Section 2.3 then summarizes the decomposition method.

2.1. Shifting barriers to obtain feasibility. Define the *shifted logarithmic barrier function*

$$\rho : \mathbb{R}^N \times \{\tau \in \mathbb{R} | \tau > 0\} \times \mathbb{R}^J \to \mathbb{R} \cup +\infty$$

by

$$\rho(x, \tau, \theta) := \begin{cases} -\tau \sum_{j=1}^{J} \ln(\theta_j - D_j x) & \text{if } \theta > Dx, \\ +\infty & \text{otherwise}, \end{cases}$$

and let

$$f(x, \tau, \theta) := c(x) + \rho(x, \tau, \theta)$$

denote the original objective function augmented by the shifted logarithmic barrier function. Also, define the corresponding *barrier problem* as

$$\mathcal{P}(\tau, \theta): \qquad \underset{x}{\text{minimize}}\ f(x, \tau, \theta) \quad \text{subject to } x \in \mathcal{B},$$

i.e., $\mathcal{P}(\tau, \theta)$ represents the optimization problem with parameters $\tau > 0$ and $\theta \in \mathbb{R}^J$. ρ was defined so that $\rho(\cdot, \tau, \theta)$ is a barrier function modeling the constraints $Dx < \theta$.

Thus, for $\theta = d$, we have $\operatorname{dom}\rho(\cdot,\tau,d) = \operatorname{int}\mathcal{C}$. Allowing $\theta \neq d$ has the property of "shifting" the barrier, hence the name.

Once an initial point $x^0 \in \mathcal{B}$ has been found by solving the relaxed problem (3), the parameters θ^1 and τ^1 are chosen so that $\tau^1 > 0$ and $\theta^1 > Dx^0$. This will have the effect of making $x^0 \in \mathcal{B}$ an interior point of $\operatorname{dom} f(\cdot,\tau^1,\theta^1)$. We then compute x^1 by approximately minimizing the barrier problem $\mathcal{P}(\tau^1,\theta^1)$. In general, if $Dx^i < d$, we have produced a feasible point. If not, then we choose θ^{i+1} as described below while maintaining $\tau^{i+1} = \tau^i$; then set $i \leftarrow i+1$ and do the process again. We will prove that if a point $x \in \mathcal{B} \cap \operatorname{int}\mathcal{C}$ exists, then *such a point is generated in a finite number of iterations*, under appropriate assumptions given below in Theorem 2.4.

Suppose first that our θ are chosen so that

$$Dx^i < \theta^{i+1} \leq \theta^i \quad \text{and} \quad \theta^i \geq d. \tag{4}$$

This implies that $\theta^\infty := \lim_{i\to\infty}\theta^i$ is well defined. We make one more assumption on our choice of θ:

$$\text{either } \theta^\infty = d \quad \text{or} \quad \exists j \text{ such that } \liminf_i \left(\theta^i_j - D_j x^i\right) = 0. \tag{5}$$

In order to develop expressions for derivatives of ρ needed in the convergence proof, let

$$q(x,\tau,\theta) := \left(\frac{\tau}{\theta_1 - D_1 x}, \cdots, \frac{\tau}{\theta_J - D_J x} \right) \tag{6}$$

and

$$Q(x,\tau,\theta) := \begin{pmatrix} \dfrac{\tau}{(\theta_1 - D_1 x)^2} & 0 & \cdots & 0 \\ 0 & \dfrac{\tau}{(\theta_2 - D_2 x)^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dfrac{\tau}{(\theta_J - D_J x)^2} \end{pmatrix}.$$

Provided $Dx < \theta$, we have

$$\nabla_x \rho(x,\tau,\theta) = q(x,\tau,\theta)D, \tag{7}$$

$$\nabla_{xx}\rho(x,\tau,\theta) = D^\top Q(x,\tau,\theta)D, \tag{8}$$

$$\nabla_\theta \rho(x,\tau,\theta) = -q(x,\tau,\theta), \quad \text{and} \tag{9}$$

$$\nabla_{\theta\theta}\rho(x,\tau,\theta) = Q(x,\tau,\theta). \tag{10}$$

We briefly review some definitions. A function ϕ is *antitone* if $a \geq b \Rightarrow \phi(a) \leq \phi(b)$. A function ϕ is said to be *essentially smooth* (see §26 of Rockafellar [16]) if $\operatorname{dom}\phi \neq \emptyset$, ϕ is differentiable on $\operatorname{dom}\phi$, and $\lim|\phi(x^i)| = +\infty$ for all sequences $\{x^i\} \subset \operatorname{dom}\phi$ converging to a point $\bar{x} \in \operatorname{bdy}\operatorname{dom}\phi$. We shall denote the restriction of a mapping ϕ to a set S by $\phi|_S$ and if S is a subspace of T, the quotient space of S in T is denoted by T/S. Where y is a point and S is a set, we use $y + S$ to denote $\{y + s | s \in S\}$.

THEOREM 2.1. *Let ϕ and ψ denote the mappings*

$$x \overset{\phi}{\mapsto} \rho(x,\tau,\theta) \quad \text{and} \quad \theta \overset{\psi}{\mapsto} \rho(x,\tau,\theta),$$

respectively, where $\tau > 0$. *Let* y *be any fixed vector in* $\mathbb{R}^N$ *and let* $S := y + \ker D$ *and* $S^{\div} := y + (\mathbb{R}^N / \ker D)$, *so that* S *and* $S^{\div}$ *are translated subspaces. Then the following are true:*

(i) $\operatorname{dom}\phi = \{x | Dx < \theta\}$,
(ii) ϕ *is essentially smooth,*
(iii) ϕ *is convex,*
(iv) $\phi|_S$ *is constant,*
(v) $\phi|_{S^{\div}}$ *is strictly convex,*
(vi) $\operatorname{dom}\psi = \{\theta | Dx < \theta\}$,
(vii) ψ *is essentially smooth,*
(viii) ψ *is antitone,*
(ix) ψ *is strictly convex.*

Proof. Clearly (i) and (vi) hold. Then (ii) and (vii) hold from the definition. Since $q(x, \tau, \theta) > \mathbf{0}$ whenever $Dx < \theta$ and $\tau > 0$, (9) shows (viii) to be true. Since $Q(x, \tau, \theta)$ is positive definite whenever $Dx < \theta$ and $\tau > 0$ we see from (8) and (10) that (iii) and (ix) are true. Note that ρ is constant on $S = y + \ker D$, so that (iv) holds, and we have shown all but (v).

For $x \in S^{\div} \cap \{x | Dx < \theta\}$ there is a one-to-one correspondence with $\{s | s < \theta\}$ given by $s = Dx$. (The reader may prove this by showing that the linear mapping $x \mapsto Dx$ is a bijection from $S^{\div}$ to range D.) Therefore, it suffices to show that the mapping

$$s \mapsto -\tau \sum_j \ln(\theta_j - s_j)$$

is strictly convex for $s < \theta$. This is true because its Hessian is given by $Q(x, \tau, \theta)$, which is positive definite. □

We let $\mathcal{X}(\tau, \theta) \subset \mathcal{B}$ denote the set of minimizers of $\mathcal{P}(\tau, \theta)$ as a function of $\tau > 0$ and θ. Also let $f^*(\tau, \theta) := f(x, \tau, \theta)$ for $x \in \mathcal{X}(\tau, \theta)$ be the optimal value function. In the case where $\operatorname{dom} f(\cdot, \tau, \theta) \cap \mathcal{B} = \emptyset$, we define $\mathcal{X}(\tau, \theta) = \emptyset$ and $f^*(\tau, \theta) = +\infty$. The following shows that $\mathcal{X}$ is nonempty in the other case.

THEOREM 2.2. *Suppose* $\tau > 0$ *and* $\theta \in \mathbb{R}^J$ *are fixed. If there is some point* $z \in \operatorname{dom} f(\cdot, \tau, \theta) \cap \mathcal{B}$, *then* $\mathcal{X}(\tau, \theta) \neq \emptyset$ *and* $f^*(\tau, \theta) < +\infty$.

Proof. Since $\mathcal{B}$ is compact and the level set $\{x | f(x, \tau, \theta) \leq f(z, \tau, \theta)\}$ is closed, the intersection of these two sets is compact. Since $f(\cdot, \tau, \theta)$ is a continuous function, it must attain its minimum on this compact intersection. Clearly, this minimum $f^*(\tau, \theta)$ is less than $+\infty$. □

The next result is quite easy to prove. It is stated as a lemma so that we may refer to it later.

LEMMA 2.3. *Suppose* $\tau > 0$ *and the sequences* $\{x^i\} \subset \mathbb{R}^N$ *and* $\{\theta^i\} \subset \mathbb{R}^J$ *satisfy* $\theta^i_j > D_j x^i$ *for each* $i = 0, 1, \cdots$ *and* $j = 1, \cdots, J$. *Then the following are equivalent:*

(i) $\rho(x^i, \tau, \theta^i)$ *is bounded for all* $i = 0, 1, \cdots$,
(ii) $\left\| q(x^i, \tau, \theta^i) \right\|$ *is bounded for all* $i = 0, 1, \cdots$,
(iii) $\theta^i_j - D_j x^i$ *is bounded away from* 0 *for all* $i = 0, 1, \cdots$ *and* $j = 1, \cdots, J$.

Proof. One may see from the definitions of ρ and q that (iii) is equivalent to both (i) and (ii). □

THEOREM 2.4. *Suppose* $c(\cdot)$ *is bounded from below on* $\mathcal{B}$. *Let* θ^{i+1} *be chosen to satisfy* (4) *and* (5). *Let* $x^i \in \mathcal{B}$ *be computed so that*

$$f(x^i, \tau, \theta^i) \leq f^*(\tau, \theta^i) + \beta \tag{11}$$

for a constant $\beta > 0$. If a point $z \in \mathcal{B} \cap \operatorname{int} \mathcal{C}$ exists, such a point will be found in a finite number of steps. On the other hand, if no such z exists, then $f(x^i, \tau, \theta^i) \to +\infty$.

(Note that by using a procedure for convex programming that generates lower bounds by solving linear programs, we may actually compute each x^i in a finite number of steps.)

Proof. Assume such a z exists. Then

$$f^*(\tau, \theta^i) \leq f(z, \tau, \theta^i) \leq f(z, \tau, d) < +\infty \quad \forall i,$$

where the second inequality follows from antitonicity in θ (part (viii) of Theorem 2.1). Therefore, for all i, $f(x^i, \tau, \theta^i)$ is bounded above by $\beta + f(z, \tau, d)$. By Lemma 2.3 and because $c(x^i)$ is bounded from below, $\theta_j^i - D_j x^i \geq \gamma > 0$ for all i, j. Therefore, for some finite $\hat{\imath}$, $d_j - D_j x^{\hat{\imath}} \geq \gamma/2 > 0$ for all j, and a point with $Dx^{\hat{\imath}} < d$ has been found in a finite number of iterations. If no such z exists, then (4) and (5) imply that $\theta_j^i - D_j x^i \to 0$ for at least one $j \in \{1, \cdots, J\}$. Lemma 2.3 then shows that $f(x^i, \tau, \theta^i) \to +\infty$. □

Assume from now on that the x^i are computed so that $Dx^i < \theta^i$. We will now give a particular method for computing θ that guarantees that (4) and (5) are satisfied (see Theorem 2.5). After computing $x^0 \in \mathcal{B}$, set

$$\theta_j^1 \leftarrow \begin{cases} d_j & \text{if } D_j x^0 < d_j, \\ D_j x^0 + \Delta & \text{if } D_j x^0 \geq d_j, \end{cases} \tag{12}$$

where $\Delta > 0$ is a constant. ($\Delta > 0$ places x^0 in the interior of the shifted feasible region.) In general, after computing x^i, set

$$\theta_j^{i+1} \leftarrow \begin{cases} d_j & \text{if } D_j x^i < d_j, \\ \lambda_\theta D_j x^i + (1 - \lambda_\theta)\theta_j^i & \text{if } D_j x^i \geq d_j, \end{cases} \tag{13}$$

where $\lambda_\theta \in (0, 1)$ is a constant.

THEOREM 2.5. *Suppose the x^i are computed so that $Dx^i < \theta^i$. If the θ are computed by the rules* (12) *and* (13)*, then* (4) *and* (5) *are satisfied.*

Proof. We show the result for each component j. If $D_j x^i \geq d_j$, then $\theta_j^{i+1} = \lambda_\theta D_j x^i + (1 - \lambda_\theta)\theta_j^i$. Condition (4) follows from this since $D_j x^i < \theta_j^i$ and $\lambda_\theta \in (0, 1)$. If $D_j x^i < d_j$, then $\theta_j^{\hat{\imath}} = d_j$ for all $\hat{\imath} > i$ and again condition (4) follows. If, for some finite i, $D_j x^i < d_j$, then condition (5) follows. In the other case ($D_j x^i \geq d_j$ for all i), the limit

$$\theta_j^\infty := \lim_{i \to \infty} \theta_j^i$$

is well defined since the sequence is monotonic and bounded. Then (13) shows that for any $s_j^\infty \in \operatorname{acc}\{D_j x^i\}$ we have

$$\theta_j^\infty = \lambda_\theta s_j^\infty + (1 - \lambda_\theta)\theta_j^\infty.$$

Therefore $s_j^\infty = \theta_j^\infty$, which shows that (5) holds. □

2.2. ε-Optimal solutions. In this section we assume that $\theta = d$ and that we have found some point in $\mathcal{B} \cap \operatorname{int} \mathcal{C}$. We develop a method that converges to a feasible point whose objective value is within a prespecified optimality tolerance ε. As before, we let $\mathcal{X}(\tau, d)$ denote the set of minimizers of $\mathcal{P}(\tau, d)$.

The following result (see Fiacco and McCormick [6] or McCormick [13, p. 341] for related results) gives us a useful bound on the error of $x \in \mathcal{X}(\tau, d)$ as compared with an optimal solution of (1).

THEOREM 2.6. *Suppose $\tau > 0$, $x \in \mathcal{X}(\tau, d)$, and x^* is an optimal solution of the original problem* (1). *Then*

$$c(x) - \varepsilon \leq c(x^*) \leq c(x),$$

for $\varepsilon = \tau J$, where J is the number of rows of D.

Proof. The second inequality follows from the feasibility of x and the optimality of x^*. The KKT conditions show that solving the barrier problem $\mathcal{P}(\tau, d)$ exactly gives multipliers p for the rows of A satisfying

$$r := \nabla_x f(x, \tau, d) + pA,$$

and complementary slackness

$$\max\{r_{k,n}(x_{k,n} - \ell_{k,n}), r_{k,n}(x_{k,n} - u_{k,n})\} = 0 \quad \forall k, n.$$

(The vector r is so named because it is the "reduced cost" vector.) Define $q^\top \in \mathbb{R}^J$ as in (6). We note that (x, p, q, r) is feasible for the dual of the original problem (1). The dual may be stated

$$\begin{aligned} \underset{x,p,q,r}{\text{maximize}} \quad & c(x) + p(Ax - b) + q(Dx - d) + r_+(x - \ell) + r_-(u - x) \\ & \nabla c(x) + pA + qD = r \\ \text{subject to} \quad & q \geq \mathbf{0} \\ & \max\{r_{k,n}(x_{k,n} - \ell_{k,n}), r_{k,n}(x_{k,n} - u_{k,n})\} = 0 \quad \forall k, n. \end{aligned}$$

It is well known that the objective value of the dual is a lower bound on $c(x^*)$. The feasibility of x and the complementarity of x and r show that the objective function of the dual is equal to

$$c(x) + q(Dx - d) = c(x) + \sum_{j=1}^{J} \frac{\tau(D_j x - d_j)}{d_j - D_j x} = c(x) - \tau J,$$

which completes the proof. □

It is instructive to point out the relation between the set-valued "trajectory" $\mathcal{X}(\tau, \theta)$ (as a function of $\tau > 0$) and the central trajectory in the interior point literature (see Bayer and Lagarias [1], especially Theorem 6.3). For the linear program in standard form

$$\text{minimize} \ \hat{c}^\top \hat{x} \ \text{ subject to } \ \hat{A}\hat{x} = \hat{b}; \qquad \hat{x} \geq \mathbf{0},$$

the central trajectory is the function $\hat{x}(\tau)$ $(\tau > 0)$ where

$$\hat{x}(\tau) := \arg\min \left\{ \hat{c}^\top \hat{x} - \tau \sum_{n=1}^{N} \ln(\hat{x}_n) \ \mid \ \hat{A}\hat{x} = \hat{b} \right\}.$$

RELAXED PHASE
$i \leftarrow 0$
Compute x^0 as the solution of the "relaxed" problem (3)
If we determine that $\mathcal{B} = \emptyset$
Then terminate (block constraints are infeasible)
Set θ^1 as in (12) and initialize τ^1
If $x^0 \in \operatorname{int} \mathcal{C}$ go to the REFINE PHASE
Otherwise go to the FEASIBILITY PHASE
FEASIBILITY PHASE
$i \leftarrow i + 1$
Generate x^i as an *approximate* solution of $\mathcal{P}(\tau^i, \theta^i)$
Set θ^{i+1} as in (13)
$\tau^{i+1} \leftarrow \tau^i$
If $x^i \in \mathcal{B} \cap \operatorname{int} \mathcal{C}$
Then go to the REFINE PHASE
Otherwise repeat the FEASIBILITY PHASE
REFINE PHASE
$i \leftarrow i + 1$
Generate x^i as an *approximate* solution of $\mathcal{P}(\tau^i, \theta^i)$
Set $\theta^{i+1} \leftarrow d$
$\tau^{i+1} \leftarrow \max\{\lambda_\tau \tau^i, \tau_{\text{inf}}\}$
Repeat the REFINE PHASE

FIG. 1. *The three-phase method.*

We see that both trajectories are defined as solutions of optimization problems where certain inequality constraints have been converted into barrier terms in the objective. Traditional trajectory-following methods would compute a sequence $\{x^i\}$ of approximations to the sequence $\{\mathcal{X}(\tau^i, \theta)\}$ where $\tau^i \downarrow 0$ in order to approximate the limit of the trajectory as $\tau \downarrow 0$.

Instead of letting $\tau^i \downarrow 0$, we use a sequence $\{\tau^i\}$ generated by the recurrence

$$\tau^{i+1} \leftarrow \max\{\lambda_\tau \tau^i, \tau_{\text{inf}}\}$$

where $\tau^1, \tau_{\text{inf}} > 0$ and $\lambda_\tau \in [0, 1)$. Since $\tau^i = \tau_{\text{inf}}$ for i sufficiently large, it suffices to use one step of a convergent iterative method for solving $\mathcal{P}(\tau, \theta)$ at each iteration. Thus, doing an infinite number of steps gives a sequence with $\operatorname{acc}\{x^i\} \subset \mathcal{X}(\tau_{\text{inf}}, d)$. Using Theorem 2.6, we may choose τ_{inf} a priori so that $c(x)$ differs from $c(x^*)$ by not more than a prespecified tolerance for $x \in \mathcal{X}(\tau_{\text{inf}}, d)$. Note that the sequence $\tau^i = \tau_{\text{inf}}$ for all i will suffice in theory. In practice, we begin with $\tau^1 \gg \tau_{\text{inf}}$ because $\mathcal{P}(\tau, d)$ is ill conditioned for $\tau \approx 0$. We do not explicitly enforce any criterion making x^i near to $\mathcal{X}(\tau^i, \theta)$. This trajectory-following idea may be viewed as a heuristic method for coping with ill-conditioning.

2.3. The three-phase method. The method we have been developing fits naturally into a three-phase framework. We assume that we are given the objective function c and the constraint sets $\mathcal{B}$ and $\mathcal{C}$. Also, we are given constants Δ, λ_θ, λ_τ, τ_{inf}, and τ^1 as introduced above.

The algorithm we use is described in Fig. 1. Note that both the FEASIBILITY PHASE and the REFINE PHASE contain the procedure

Generate x^i as an *approximate* solution of $\mathcal{P}(\tau^i, \theta^i)$.

We shall now discuss how this is implemented.

Note that the sequences $\{\theta^i\}$ and $\{\tau^i\}$ generated by the REFINE PHASE of this method have the properties that $\theta^i = d$ and $\tau^i = \tau_{\inf}$ for all i sufficiently large. Thus, for the REFINE PHASE, we use only *one step* of some iterative method for convex programming to generate x^{i+1} from x^i. The resulting sequence $\{x^i\}$ will then have $\operatorname{acc}\{x^i\} \subset \mathcal{X}(\tau_{\inf}, d)$. For the FEASIBILITY PHASE, one step *may* suffice, but the hypotheses of Theorem 2.4 (finite feasibility) would not necessarily hold.

3. Solving barrier problems approximately. We assume in this section that $\tau > 0$ and θ are given, and that at the current iteration i we are given $x^i \in \mathcal{B}$ such that $f(x^i, \tau, \theta)$ is finite. Our method for obtaining approximate solutions of barrier problems consists of approximating the objective function to allow the *computation of search directions separately for each block*, and then *coordinating* them. The method we use is a generalization of the Frank–Wolfe method that uses a trust region and takes advantage of the block structure of the constraints by using a multi-dimensional search rather than a line search.

3.1. Linearizing to obtain block search directions in parallel. Suppose we are given a current point $x^i \in \mathcal{B}$. Linearizing $\mathcal{P}(\tau, \theta)$ and adding a trust region $R(x^i)$ gives the following problem:

$$\underset{y}{\text{minimize}}\, \nabla_x f(x^i, \tau, \theta) y \quad \text{subject to } y \in \mathcal{B} \cap R(x^i), \tag{14}$$

which is of the form (2) with $\tilde{c} = \nabla_x f(x^i, \tau, \theta)$ and $R(x^i) = \{y | \tilde{\ell} \le y \le \tilde{u}\}$. Let y^i denote the solution of (14) and define the block search direction as $\delta y^i := y^i - x^i$. Recall our assumption that (14) is easy to solve and may be solved *in parallel by taking advantage of block structure and decomposing into K independent subproblems.*

The purpose of the trust region is to take into account the poles of the barrier function, where the logarithm is undefined. We use the following choice for the trust region $R(\cdot)$. Recall that the objective function is

$$f(x^i + \delta y, \tau, \theta) = c(x^i + \delta y) - \tau \sum_j \ln\left(\theta_j - D_j(x^i + \delta y)\right),$$

from which we immediately see that

$$x^i + \delta y \in \operatorname{dom} f \iff (\forall j) \qquad D_j \delta y < \theta_j - D_j x^i. \tag{15}$$

This shows that enforcing $x^i + \delta y \in \mathcal{B} \cap \operatorname{dom} f$ would involve essentially the same constraints as the original problem (1). Instead, we propose approximating condition (15) by modification of the simple bounds that already occur in the constraints $x + \delta y \in \mathcal{B}$. To this end, suppose we were to let only one component (the (k, n) component, say) of δy be nonzero. Then $x^i + \delta y \in \operatorname{dom} f$ only if $D_{j,k,n} \delta y_{k,n} < \theta_j - D_j x^i$. Therefore, we use the "one-sided" trust region R defined by

$$R(x^i) := \left\{ x = x^i + \delta y | D_{j,k,n} \delta y_{k,n} \le \max\left\{\hat{\delta}, \gamma(\theta_j - D_j x^i)\right\} \quad \forall j, k, n \right\} \tag{16}$$

where $\hat{\delta} > 0$ and $\gamma \in (0, 1]$ are constants (see Fig. 2); $\hat{\delta} > 0$ guarantees that the distance to the boundary of the trust region is bounded away from 0. Such a trust region does not guarantee that $x^i + \delta y \in \operatorname{dom} f$, but tends to limit extreme violation of this "domain constraint." Enforcement of "domain constraints" is ensured by the coordinator process, discussed below.

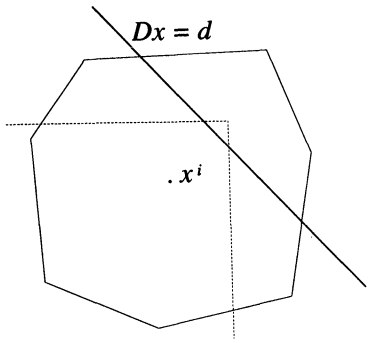


FIG. 2. *A typical trust region ($\mathcal{B}$ is represented by the solid polygon).*

More generally, the trust region property that we require in the convergence proof is the following.

ASSUMPTION 3.1. *Let $z \in \mathcal{B}$ and the sequence $\{x^i\} \subset \mathcal{B}$, and define*

$$\alpha^i := \max_{0 \le \alpha \le 1} \{\alpha | x^i + \alpha(z - x^i) \in R(x^i)\}.$$

Then $\liminf_{i \to \infty} \alpha^i > 0$.

If this assumption is satisfied, there is always some room to move in any direction from x^i without being constrained by the trust region. Assumption 3.1 is clearly true for our choice of R in (16) because $\hat{\delta} > 0$ and $\mathcal{B}$ is bounded.

3.2. Coordination. The Frank–Wolfe method would use a one-dimensional search along the line segment $\text{conv}\{x^i, y^i\}$ from x^i to y^i. If we search over a larger region we may expect to do better. The block structure of the original problem makes a multidimensional search quite natural. More precisely, we will show that the block structure of A means that we may solve a K-dimensional optimization problem with simple bounds in order to "coordinate" the subproblem solutions. In this section, $f(\cdot)$ and $\nabla f(\cdot)$ are used in place of $f(\cdot, \tau, \theta)$ and $\nabla_x f(\cdot, \tau, \theta)$ since τ and θ are fixed.

Let Y^i denote the $N \times K$ matrix of block search directions:

$$Y^i := \begin{pmatrix} \delta y_1^i & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \delta y_2^i & \cdots & \mathbf{0} \\ & & & \\ \mathbf{0} & \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \delta y_K^i \end{pmatrix}$$

so that the kth column of Y^i corresponds to the kth block of the update δy^i. The coordination problem that we consider generates w^i as an approximate solution of

$$\underset{w}{\text{minimize}}\, f(x^i + Y^i w) \quad \text{subject to } \ell \le x^i + Y^i w \le u. \tag{17}$$

We will then choose $x^{i+1} = x^i + Y^i w^i$ for our next iterate. Note that $x^{i+1} \in \mathcal{B}$ if $x^i \in \mathcal{B}$ because Y^i is in the null-space of the equations defining $\mathcal{B}$. Also, it is easy to enforce $x^{i+1} \in \text{dom}\, f$ since $x^i \in \text{dom}\, f$. Our approximate solutions of (17) will satisfy

$$f(x^{i+1}) - f(x^i) \le \mu \left[\min_{x \in \text{conv}\{x^i, y^i\}} f(x) - f(x^i) \right] \tag{18}$$

where $\mu \in (0,1)$ and where the minimization is taken over the line segment from x^i to y^i. (We consider below specific methods for guaranteeing satisfaction of (18) in a finite number of iterations.)

We now show that this method converges to solutions of $\mathcal{P}(\tau,\theta)$ where τ and θ are fixed. From (18) we may deduce that the algorithm is monotonic, i.e., $f(x^{i+1}) \leq f(x^i)$. Consider a subsequence $\{x^{\sigma(i)}\} \subset \{x^i\}$ that has $x^{\sigma(i)} \to \bar{x}$ and $y^{\sigma(i)} \to \bar{y}$. This subsequence must exist because $\mathcal{B}$ is compact. Also define $\overline{\delta y} := \bar{y} - \bar{x}$ so that $\delta y^{\sigma(i)} \to \overline{\delta y}$. Pick any $\lambda \in (0,1]$. Then

$$\begin{aligned} f(x^{\sigma(i)} + \lambda \delta y^{\sigma(i)}) - f(x^{\sigma(i)}) &\geq \left[\min_{x \in \operatorname{conv}\{x^{\sigma(i)}, y^{\sigma(i)}\}} f(x) - f(x^{\sigma(i)})\right] \\ &\geq \frac{1}{\mu}\left[f(x^{\sigma(i)+1}) - f(x^{\sigma(i)})\right]. \end{aligned}$$

Since $f(\cdot)$ is bounded from below on $\mathcal{B}$, taking the limit as $i \to \infty$ gives $f(\bar{x} + \lambda\overline{\delta y}) - f(\bar{x}) \geq 0$, and taking the limit of this as $\lambda \downarrow 0$ gives

$$0 \leq \nabla f(\bar{x})\overline{\delta y} = \nabla f(\bar{x})\,(\bar{y} - \bar{x})\,. \tag{19}$$

Suppose now that $z \in \mathcal{B}$ is an arbitrary point. Since R satisfies Assumption 3.1,

$$\alpha^i = \max_{0 \leq \alpha \leq 1} \{\alpha | x^i + \alpha(z - x^i) \in R(x^i)\}$$

has $\liminf \alpha^i > 0$. Then

$$\nabla f(x^i)(y^i - x^i) \leq \nabla f(x^i)\alpha^i(z - x^i)$$

by the definition of y^i. By taking the limit, and using (19) and the convexity of f,

$$0 \leq \nabla f(\bar{x})(z - \bar{x}) \leq f(z) - f(\bar{x}).$$

This shows that $\bar{x}$ is a global minimizer of f over $\mathcal{B}$ and hence $\bar{x} \in \mathcal{X}(\tau,\theta)$.

Except in special cases, the minimum in condition (18) may not be computed in a finite number of steps. We therefore substitute a computable lower bound.

Let $\phi(w) = f(x^i + Y^i w)$, so that the coordinator problem (17) may be stated

$$\underset{w}{\text{minimize}}\, \phi(w) \quad \text{subject to } \underline{w} \leq w \leq \overline{w} \tag{20}$$

for bounds $\underline{w}$ and $\overline{w}$ that correspond to the bounds of (17).[1] (Note that $\underline{w} \leq \mathbf{0} < \mathbf{1} \leq \overline{w}$ because x^i and $x^i + y^i$ satisfy the bound constraints.) Since ϕ is convex,

$$\phi(\hat{w}) + \nabla\phi(\hat{w})(w - \hat{w}) \leq \phi(w)$$

for all w. Restricting the linearization on the left to $w \in \operatorname{conv}\{\mathbf{0},\mathbf{1}\}$ (which corresponds to $x \in \operatorname{conv}\{x^i, y^i\}$ in the original space) we will obtain a lower bound on the minimum in (18). Clearly this linearization restricted to a line segment will achieve its minimum at an endpoint. Therefore, defining

$$\psi(\hat{w}) := \phi(\hat{w}) + \min\{\nabla\phi(\hat{w})(\mathbf{0} - \hat{w}), \nabla\phi(\hat{w})(\mathbf{1} - \hat{w})\},$$

[1] The components $\underline{w}_k$ and $\overline{w}_k$ are computed by looking at all ratios of the form $\left(\ell_{k,n} - x^i_{k,n}\right)/\delta y^i_{k,n}$ and $\left(u_{k,n} - x^i_{k,n}\right)/\delta y^i_{k,n}$ in order to obtain the box $\{w | \underline{w} \leq w \leq \overline{w}\} \subset \{w | \ell \leq x^i + Y^i w \leq u\}$ that is the largest possible.

we have shown

$$\psi(\hat{w}) \leq \min_{w \in \text{conv}\{\mathbf{0},\mathbf{1}\}} \phi(w) = \min_{x \in \text{conv}\{x^i, y^i\}} f(x).$$

Thus, instead of satisfying (18) directly, we instead require

$$\phi(w^i) - \phi(\mathbf{0}) \leq \mu \left[\psi(w^i) - \phi(\mathbf{0})\right], \tag{21}$$

in order to accept w^i. A w^i satisfying (21) may, therefore, be computed in a finite number of iterations for $\mu < 1$. Then setting $x^{i+1} \leftarrow x^i + Y^i w^i$ implies that (18) is satisfied.

We may define the "reduced gradient" of ϕ at w componentwise as

$$(\text{rg}\,\phi(w))_k := \begin{cases} [(\nabla\phi(w))_k]_+ & \text{if } w_k = \underline{w}_k, \\ -[(\nabla\phi(w))_k]_- & \text{if } w_k = \overline{w}_k, \\ (\nabla\phi(w))_k & \text{otherwise .} \end{cases}$$

In practice, we have found that it is better to choose w^i to satisfy both (21) and

$$\left\|\text{rg}\,\phi(w^i)\right\| \leq \mu' \left\|\text{rg}\,\phi(\mathbf{0})\right\|$$

(in some norm $\|\cdot\|$) for some $\mu' \in (0,1)$. That is, we accept w^i as a solution to the coordinator problem if it has sufficient decrease to ensure convergence (i.e., (21) is satisfied) *and* the reduced gradient is sufficiently reduced in norm.

The coordinator problem may be enriched by utilizing one or more sets of "prior" updates as well as the "current" updates Y^i. Since all updates lie in the null space of the equality constraints of $\mathcal{B}$, we need only take into account the bounds $\ell \leq x \leq u$. We shall discuss the case where our prior updates are merely the set of updates Y^{i-1} from the previous iteration. Let v be the vector of weights for the prior update Y^{i-1}, analogous to the role of w for the current update Y^i. The coordination constraints are then

$$\ell \leq x^i + Y^{i-1}v + Y^i w \leq u, \tag{22}$$

which no longer reduce to simple bounds on v and w. However, we may easily compute sets of separate bounds on v and w that form an "inner approximation" of (22) in the sense that any (v,w) feasible for the bounds must also be feasible for (22). To do this, choose a constant $\xi \in (0,1)$ and constrain w by $\xi\underline{w} \leq w \leq \xi\overline{w}$, where $\underline{w}$ and $\overline{w}$ are obtained exactly as above. Having set these constraints, it is easy to compute $\underline{v}$ and $\overline{v}$ such that

$$[\xi\underline{w} \leq w \leq \xi\overline{w} \text{ and } \underline{v} \leq v \leq \overline{v}] \Rightarrow (v,w) \quad \text{satisfies (22).}$$

The convergence proof is extended in a straightforward manner to cover generalized coordination procedures of this type. This approach is related to the PARTAN method of combining update information (see Himmelblau [10] and Lee et al. [12]), and also to restricted simplicial decomposition (see von Hohenbalken [11], Mulvey, Zenios, and Anlfeld [14] and Hearn, Lawphongpanich, and Ventura [9]).

4. Numerical results for a large-scale application. This section describes our test problems and documents the results produced by our decomposition algorithm. Section 4.1 describes our code and the key parameters used. In §4.2 we describe the problems that motivated this decomposition algorithm. Finally, §4.3 shows the timing results on the large-scale problems.

4.1. Description of the code and parameters. We ran our code on two machines: a DECstation 3100 running the ULTRIX operating system, and a 20 processor Sequent Symmetry S81 running the DYNIX operating system. Most of the code was written in C, with the portion used to solve the network subproblems being written in FORTRAN (see further discussion below). The C programming language was chosen primarily because of its ability to work properly with modern data structures. The code was compiled with the default code optimization (`-O1`). Double precision was used for all calculations.

The following parameter values were used in our runs (see §2):

- $\Delta = 1$: Assumes right-hand side of moderate size ($d_j \in [-10^6, 10^6]$).
- $\lambda_\theta = 0.9$: We found that if this parameter is smaller, the method does not find feasible solutions as quickly. Larger values of λ_θ do not seem to cause numerical problems for our test problems.
- $\lambda_\tau = 0.5$: Smaller values tend to introduce the problems of ill-conditioning earlier, so that convergence is hampered. Larger values require too many iterations until τ^i is sufficiently small.
- $\tau^1 = 10$: We normalize the cost coefficients so that $\|c\|_\infty = 1$, making τ^1 ten times the maximum absolute value of the cost coefficients.
- $\varepsilon = \tau_{\inf} J = 10^{-8}$: (See Theorem 2.6.) Eight places of accuracy in the objective function is a fairly ambitious goal for problems as large as our test problems. This was not always achieved in 50 iterations.

The code takes full advantage of the network structure of the A_k, since solving multicommodity networks was the initial goal of this work. The code also takes advantage of the special structure of the matrix D. Typically, for multicommodity networks, a constraint $D_j x \leq d_j$ represents a physical situation where the flow of a given "topological arc" (an arc appearing at most once in each commodity) can only handle a certain capacity of flow. In this case $D_{j,k,n} \in \{0, 1\}$, and for each j and k, there is at most one $\hat{n}$ such that $D_{j,k,\hat{n}} = 1$. Our code has a $J \times K$ array of pointers to this $\hat{n}$ (it stores 0 if no such $\hat{n}$ exists). This saves space because J and K are usually much smaller than N.

To solve the network subproblems we use a modified version of RNET, written in FORTRAN by Grigoriadis and Hsu [8]. RNET is an implementation of the network simplex method. This code was modified by the authors to work in double precision rather than integer arithmetic, and to use parameters to specify input data. We begin RNET with an all-artificial basis at each iteration. The parameters given to RNET are mostly determined by the suggestions in Grigoriadis and Hsu [8]. We allow for a large number of pivots.

For the trust region in §3.1 we set $\gamma = 0.7$ and $\hat{\delta} = 10^{-8}$. We found that $\gamma \in [0.5, 0.9]$ worked reasonably well. We also found, somewhat to our surprise, that using smaller $\hat{\delta}$ seemed to make the algorithm perform better. Using a smaller $\hat{\delta}$ means that we let the current point get closer to the boundary of the trust region, possibly at the expense of being able to move less within the null space of D. We found that $\hat{\delta} = 0$ (in which case our convergence proof fails) worked just fine in practice.

The coordinator algorithm uses an active set method in conjunction with the steepest descent direction. We stop when both the function and the norm of the projected gradient have been sufficiently decreased, as is explained at the end of §3.2. The code uses $\mu = 0.4$ and $\mu' = 0.03$. If we stop when the function values have been sufficiently decreased, but ignore the projected gradient, then the method converges in theory, but in practice it seems somewhat problematic. Using a larger μ' would

allow the algorithm to terminate, when in fact the coordinator could probably find a significantly better point at low cost. We run the coordinator algorithm for at most 15 iterations within each major iteration.

As part of this coordinator method we need to use a one-dimensional line search method. The one-dimensional line search algorithm we used is (2.6.4) in Fletcher [7]. Special structure of the objective function allows us to use Newton's method in place of the usual minimization of a quadratic or cubic interpolant. Parameters were set to attain a line search of medium accuracy.

4.2. Patient Distribution System (PDS) problems. The test problems we used were obtained from the CINCMAC analysis group of the Military Airlift Command (MAC) at Scott Air Force Base (Chmielewski [4]). The model is called the Patient Distribution System (PDS) and is a logistics model designed to help make decisions about how well MAC can evacuate patients from Europe. The PDS problems are a class of problems: PDS-$\mathcal{D}$ denoting the problem that models a scenario lasting $\mathcal{D}$ days, for integers $\mathcal{D} \in [1, 85]$. As $\mathcal{D}$ becomes larger, the size of PDS-$\mathcal{D}$ grows quite large, as may be seen in Table 1. The PDS problems are linear multicommodity network flow problems, which are block angular linear programs where each A_k is a node-arc-incidence matrix. (See Fig. 3.)

These PDS problems have received considerable attention lately, partly because they are a real-world application, and partly because they seem to be quite challenging. Carolan et al. [2] used the KORBX system at Scott Air Force Base to solve numerous problems, including some of the smaller PDS problems. It took the KORBX system (using default parameters) between 3.3 hours and 4.5 hours to solve PDS-10. Only one out of the four KORBX codes finished within 24 hours on PDS-20. Setiono [18] has solved small- and medium-sized PDS problems using a dual proximal point linear programming algorithm, solving the resulting linear systems with the preconditioned conjugate gradient method. Setiono solved PDS-20 in 25.5 hours on an Astronautics ZS-1 computer [19].[2] Using a decomposition technique, Pinar and Zenios [15] solved many of the PDS problems. The largest problem they report on is PDS-30, taking slightly more than two hours to solve on a CRAY Y-MP. Cheng et al. [3] report the solution of problems as large as PDS-50 in 10.2 hours using the KORBX system (apparently not the same version as Carolan et al. [2] use). In an unpublished work, De Leone [5] has solved PDS-40 using an SOR-based technique in approximately 27 hours on a DECstation 3100.

4.3. Results of numerical experiments. We shall now present the performance results of our codes on a subset of the PDS problems. We were interested in two things when beginning these tests. First, we wanted to develop algorithms that compute approximate solutions to multicommodity network flow problems quickly. Second, we want our method to compute accurate solutions. Although our solution is primal feasible (always $\ell \leq x \leq u$, $Dx < d$, and $\|Ax - b\|_\infty / \|b\|_\infty \approx$ machine epsilon), our method does not provide good bounds on the gap between the objective value obtained and the optimal value. We compute duals on the network constraints and the dual estimates q (defined in (6)), and these will give lower bounds as in the proof of Theorem 2.6. When we have computed these lower bounds, however, they are not even as good as the lower bound given by solving the relaxed problem (3). However, when we compare our final objective values with those obtained by others,

[2] For reference we note that Smith and Klinger [19] claim that the ZS-1 achieves 3.0 Mflops on the 24 Livermore loops and 6.3 Mflops on the 100×100 Linpack benchmark.

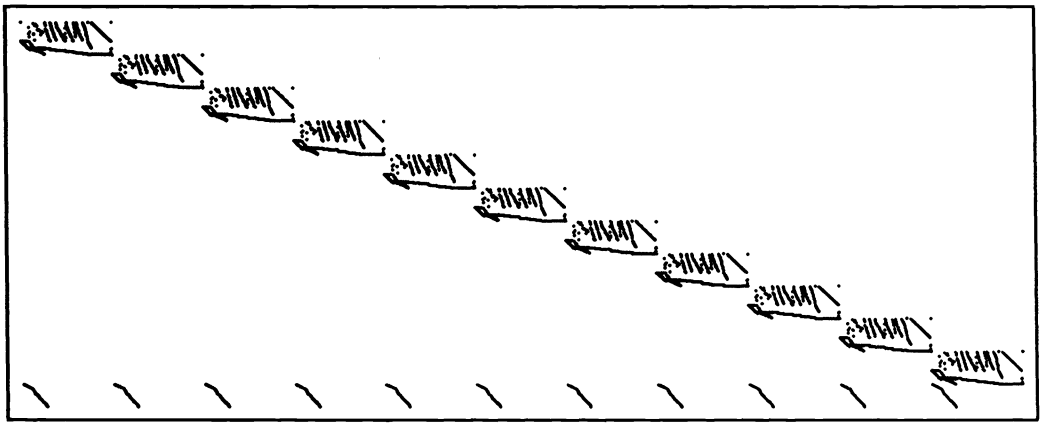


FIG. 3. *Sparsity structure of the constraint matrix for* PDS-01.

TABLE 1

Sizes of some of the PDS *problems. We also remind the reader that each* PDS *problem has eleven blocks (i.e.,* $K = 11$).

Problem Name	max size of block		Coupling	Dimension of A	
	$\max_k M(k)$	$\max_k N(k)$	J	M	N
PDS-01	126	339	87	1,386	3,729
PDS-02	252	685	181	2,772	7,535
PDS-03	390	1117	303	4,290	12,287
PDS-05	686	2,149	553	7,546	23,639
PDS-06	835	2,605	696	9,185	28,655
PDS-10	1,399	4,433	1,169	15,389	48,763
PDS-20	2,857	10,116	2,447	31,427	105,728
PDS-30	4,223	15,126	3,491	46,453	154,998
PDS-40	5,652	20,698	4,672	62,172	212,859
PDS-50	7,031	26,034	5,719	77,341	270,095
PDS-60	8,423	31,474	6,778	92,653	329,643
PDS-70	9,750	36,262	7,694	107,250	382,311
PDS-80	10,989	40,259	8,302	120,879	426,278

they typically match to between five and seven significant figures.

We ran the code for 50 iterations in all cases. Figures 4, 6, and 8 show the original objective functions $c^\top x^i$ as a function of iteration i for three of the larger PDS problems. Note that the objective function increases during the FEASIBILITY PHASE and then decreases steadily during the REFINE PHASE. Figures 5, 7, and 9 graph the improvement

$$\kappa^i := \begin{cases} -\log_{10}\left(\frac{c^\top x^{i-1} - c^\top x^i}{|c^\top x^i|}\right) & \text{if } c^\top x^{i-1} > c^\top x^i, \\ \text{undefined} & \text{otherwise} \end{cases}$$

as a function of the iteration i for the same three problems. (One might say that "the κ^ith digit of the objective function changed from iteration $i-1$ to iteration i.") These figures show that the improvement in the objective function $c^\top x^i$ tends to become small after 50 iterations. The large improvement for PDS-60 in iteration 49 illustrates

that convergence is not always predictable from objective functions alone. The primal objective function values we obtained matched known good approximations of optimal values to between five and seven digits [5], [18].

Tables 2 and 3 contain timings and optimal objective function values for the PDS problems we solved. The column of the table labeled "relaxed" contains statistics for computing the solution to (3). The column labeled "feasible" contains statistics for computing a feasible point. The number of iterations required to obtain a feasible solution via the shifted barrier approach varied between 11 and 16. The column labeled "Final" contains statistics for computing the final approximation of the optimal solution. The row of the table labeled "Total iterations" is the number of iterations the method has taken to attain a given phase. The row labeled "Objective $\times 10^{-10}$" is the value of $10^{-10} c^\top x^i$ where x^i is the current point and c is the *original* cost vector ($\|c\|_\infty$ is *not* necessarily 1). A row labeled "DECstation" shows the performance on the DECstation 3100 . A row labeled "Sequent($\wp$)" shows the performance on the Sequent Symmetry using $\wp$ processors. All times reported are wall clock time.

The version of the code on the Sequent Symmetry uses the most obvious parallel strategy; at each iteration a separate subproblem for each commodity is solved on a separate processor. While there are other possibilities for parallelism in the program (e.g., parallel function evaluation in the coordinator or overlapping coordinator and subproblems in a chaotic fashion [17]), most of the work is done in solving the (large scale) subproblems. Speedups of 4 or 5 are typical with 11 processors (and 11 commodities). This corresponds to perfect (linear) speedup of the subproblem solutions if between 80 percent and 90 percent of the work is done in solving the subproblems.

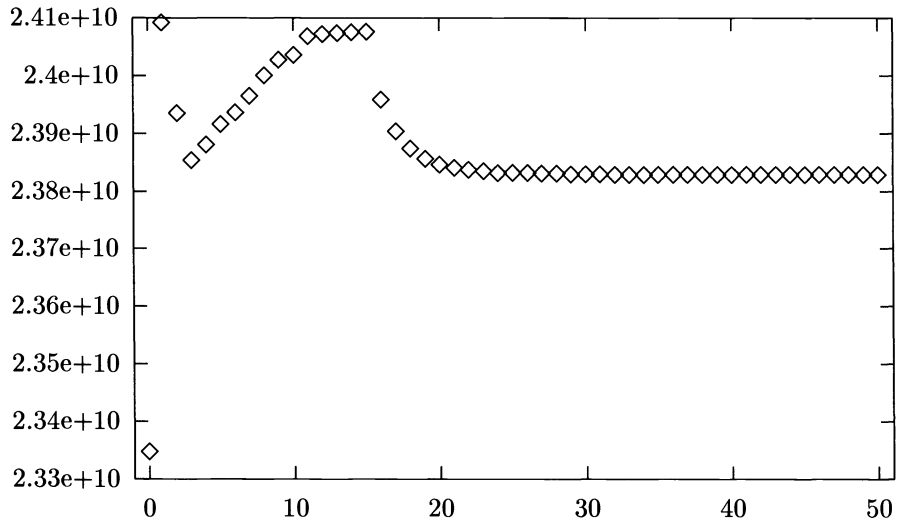


FIG. 4. PDS-20: *objective function vs. iterations.*

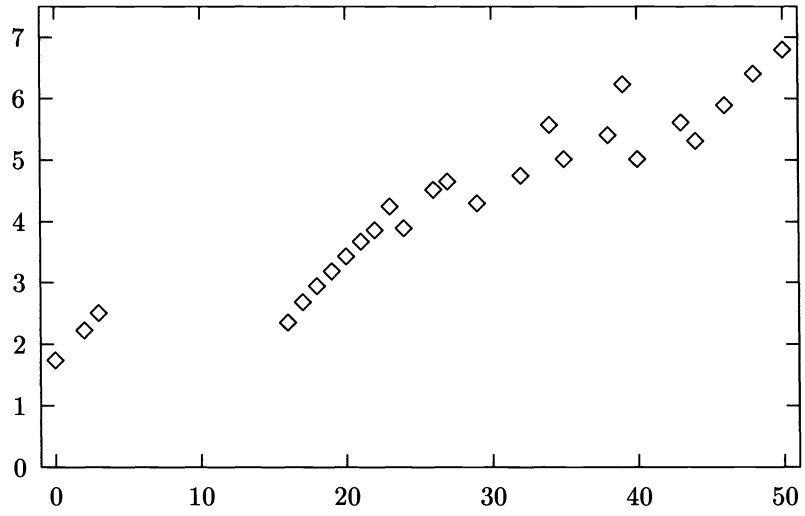


FIG. 5. PDS-20: *−log of improvement* (κ) *vs. iterations.*

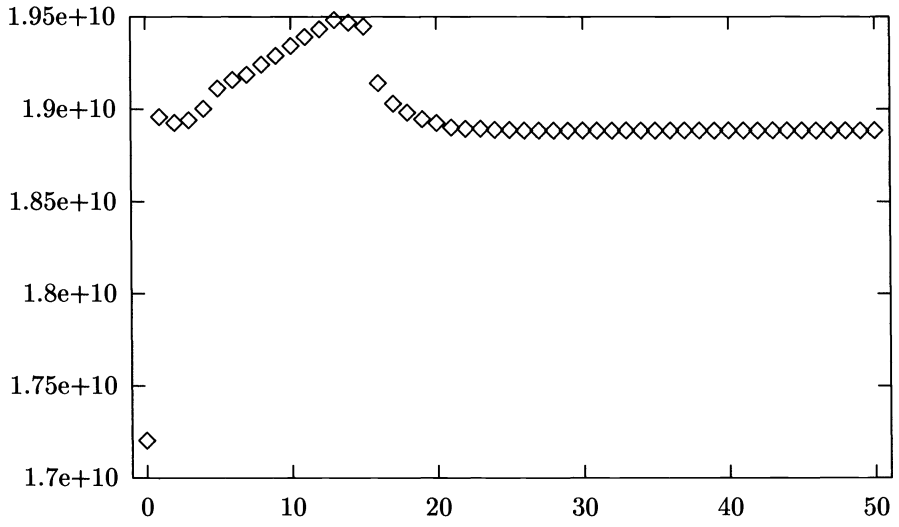


FIG. 6. PDS-40: *objective function vs. iterations.*

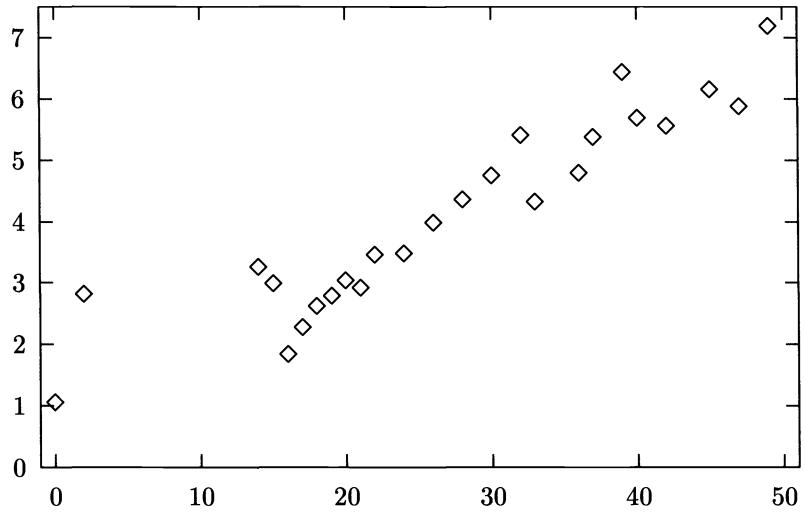


FIG. 7. PDS-40: *−log of improvement* (κ) *vs. iterations.*

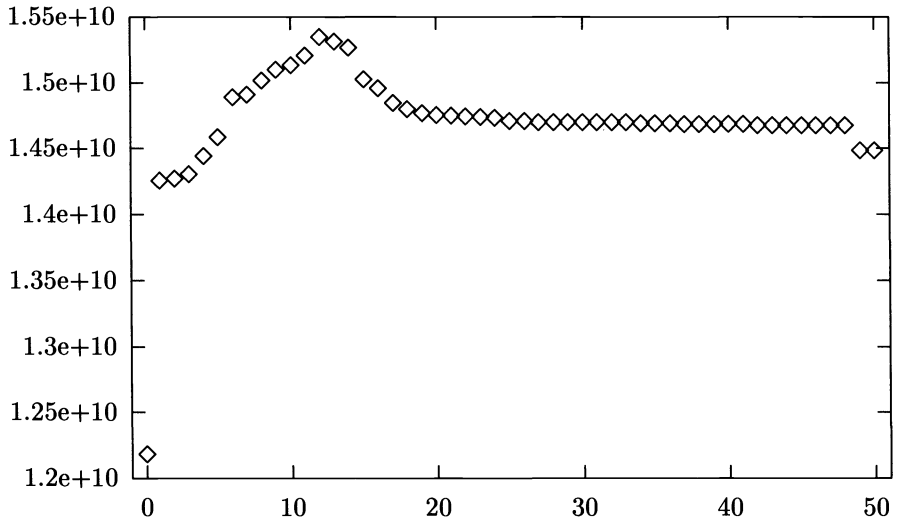


FIG. 8. PDS-60: *objective function vs. iterations.*

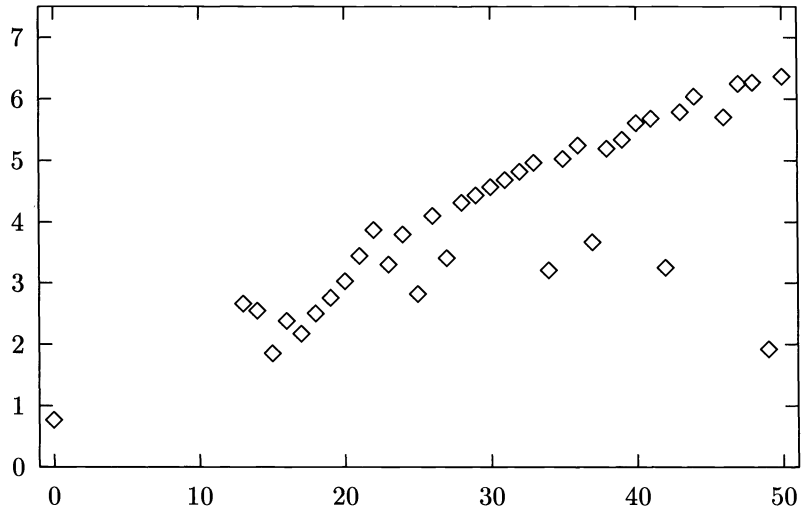


FIG. 9. PDS-60: *−log of improvement* (κ) *vs. iterations.*

TABLE 1.
Timing and objective value results for small PDS *problems.*

PDS-01			
Phase	Relaxed	Feasible	Final
Total iterations	0	11	50
Objective $\times 10^{-10}$	2.9033	2.9096	2.9084
DECstation	1.4sec	18sec	1min 30sec
Sequent(11)	1.1sec	12sec	1min 4sec
PDS-02			
Phase	Relaxed	Feasible	Final
Total iterations	0	12	50
Objective $\times 10^{-10}$	2.8758	2.8876	2.8858
DECstation	2.9sec	42sec	3min 17sec
Sequent(11)	1.5sec	22sec	2min 9sec
PDS-03			
Phase	Relaxed	Feasible	Final
Total iterations	0	13	50
Objective $\times 10^{-10}$	2.8442	2.8622	2.8597
DECstation	4.7sec	1min 20sec	5min 47sec
Sequent(11)	2sec	39sec	3min 38sec
PDS-05			
Phase	Relaxed	Feasible	Final
Total iterations	0	16	50
Objective $\times 10^{-10}$	2.7824	2.8125	2.8054
DECstation	15sec	3min 14sec	11min 28sec
Sequent(11)	4.2sec	1min 33sec	6min 43sec
PDS-06			
Phase	Relaxed	Feasible	Final
Total iterations	0	16	50
Objective $\times 10^{-10}$	2.7526	2.7846	2.7761
DECstation	17sec	4min 17sec	15min 4sec
Sequent(11)	4.6sec	1min 56sec	8min 44sec

TABLE 2
Timing and objective value results for large PDS *problems.*

PDS-10			
Phase	Relaxed	Feasible	Final
Total iterations	0	16	50
Objective $\times 10^{-10}$	2.6333	2.6857	2.6727
DECstation	30sec	8min 15sec	28min 31sec
Sequent(11)	9sec	3min 57sec	16min 39sec
PDS-20			
Phase	Relaxed	Feasible	Final
Total iterations	0	14	50
Objective $\times 10^{-10}$	2.3342	2.4069	2.3822
DECstation	2min 22sec	33min 19sec	2hr 12min
Sequent(11)	40sec	9min 44sec	50min 43sec
PDS-30			
Phase	Relaxed	Feasible	Final
Total iterations	0	12	50
Objective $\times 10^{-10}$	2.0284	2.1818	2.1390
DECstation	4min 38sec	1hr 9min	5hr 23min
Sequent(11)	1min 40sec	16min 43sec	1hr 48min
PDS-40			
Phase	Relaxed	Feasible	Final
Total iterations	0	14	50
Objective $\times 10^{-10}$	1.7188	1.9452	1.8866
DECstation	8min 53sec	2hr 39min	10hr 27min
Sequent(11)	3min 45sec	32min 32sec	2hr 54min
PDS-50			
Phase	Relaxed	Feasible	Final
Total iterations	0	13	50
Objective $\times 10^{-10}$	1.5002	1.7336	1.6625
DECstation	13min 28sec	3hr 50min	16hr 46min
Sequent(11)	4min 35sec	54min 2sec	5hr 30min
PDS-60			
Phase	Relaxed	Feasible	Final
Total iterations	0	13	50
Objective $\times 10^{-10}$	1.2159	1.5288	1.4462
DECstation	18min 40sec	5hr 27min	24hr 6min
Sequent(11)	5min 38sec	1hr 19min	6hr 55min
PDS-70			
Phase	Relaxed	Feasible	Final
Total iterations	0	16	50
Objective $\times 10^{-10}$	0.9309	1.3191	1.2311
Sequent(11)	8min 12sec	2hr 9min	9hr 24min

5. Summary and conclusions. We have developed an interior point method for block angular problems. This three-phase approach takes advantage of the constraint structure by keeping the block-structured constraints explicitly and using barrier functions to model the coupling constraints. We obtain a starting point for the shifted barrier approach by relaxing all coupling constraints. In computational experience with large-scale PDS problems, this shifted barrier approach has produced feasible points in a small number of iterations. The block angular structure allows us to compute search directions for a coordination process quickly and in parallel. We also discussed techniques that assure convergence to an ε-optimal point. This algorithm is particularly well suited to multicommodity flow problems, since the subproblems are then linear single-commodity networks. Computational experience with a class of real-world multicommodity problems arising from an Air Force MAC application indicates that the method is very efficient, even for problems with hundreds of thousands of variables.

REFERENCES

[1] D. BAYER AND J. LAGARIAS, *The nonlinear geometry of linear programming.* II *Legendre transform coordinates and central trajectories*, Trans. Amer. Math. Soc., 314 (1989), pp. 527–581.

[2] W. CAROLAN, J. HILL, J. KENNINGTON, S. NIEMI, AND S. WICHMANN, *An empirical evaluation of the* KORBX *algorithms for military airlift applications*, Oper. Res., 38 (1990), pp. 240–248.

[3] Y.-C. CHENG, D. HOUCK, JR., J.-M. LIU, M. MEKETON, L. SLUTSMAN, R. VANDERBEI, AND P. WANG, *The* AT&T KORBX^{TM} *system*, AT&T Tech. J., 68 (1989), pp. 7–19.

[4] R. CHMIELEWSKI, Private communication, March 1989.

[5] R. DE LEONE, Private communication, 1990.

[6] A. FIACCO AND G. MCCORMICK, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons, New York, 1968.

[7] R. FLETCHER, *Practical Methods of Optimization*, Second Edition, John Wiley and Sons, New York, 1987.

[8] M. GRIGORIADIS AND TAU HSU, RNET, *The Rutgers Minimum Cost Network Flow Subroutines, Users' Documentation*, 3.6 Edition, Department of Computer Science, Hill Center for the Mathematical Sciences, Rutgers University, New Brunswick, NJ, October 1979.

[9] D. HEARN, S. LAWPHONGPANICH, AND J. VENTURA, *Restricted simplicial decomposition: computation and extensions*, Math. Programming Stud., 31 (1987), pp. 99–118.

[10] D. HIMMELBLAU, *Applied Nonlinear Programming*, McGraw–Hill, New York, 1972.

[11] B. VON HOHENBALKEN, *Simplicial decomposition in nonlinear programming algorithms*, Math. Programming, 13 (1977), pp. 49–68.

[12] D. LEE, K. MEDHI, J. STRAND, R. COX, AND S. CHEN, *Solving large telecommunications network loading problems*, AT&T Tech. J., 68 (1989), pp. 48–56.

[13] G. MCCORMICK, *Nonlinear Programming, Theory, Algorithms, and Applications*, John Wiley and Sons, New York, 1983.

[14] J. MULVEY, S. ZENIOS, AND D. ANLFELD, *Simplicial decomposition for convex generalized networks*, J. Inform. Optim. Sciences, 11 (1990), pp. 359–387.

[15] M. PINAR AND S. ZENIOS, *Parallel decomposition of multicommodity network flows using a linear-quadratic penalty algorithm*, Tech. Report 90-12-06, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, December 1990.

[16] R. ROCKAFELLAR, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1970.

[17] G. SCHULTZ AND R. MEYER, *A flexible parallel algorithm for block-constrained optimization problems*, in Impacts of Recent Computer Advances on Operations Research, R. Sharda, B. Golden, E. Wasil, O. Balci, and W. Stewart, eds., North Holland, New York, 1989.

[18] R. SETIONO, *Interior dual proximal point algorithm using preconditioned conjugate gradient*, Tech. Report 951, Computer Sciences Department, University of Wisconsin, Madison, WI, July 1990.

[19] J. SMITH AND S. KLINGER, *Performance of the Astronautics* ZS-1 *central processor*, Tech. Report, Astronautics Corporation of America, 5800 Cottage Grove Rd., Madison, WI 53716, March 1988.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 603–619, November 1991

011

ON THE RATE OF CONVERGENCE OF A PARTIALLY ASYNCHRONOUS GRADIENT PROJECTION ALGORITHM*

PAUL TSENG†

Abstract. Recently, Bertsekas and Tsitsiklis proposed a partially asynchronous implementation of the gradient projection algorithm of Goldstein and Levitin and Polyak for the problem of minimizing a differentiable function over a closed convex set. In this paper, the rate of convergence of this algorithm is analyzed. It is shown that if the standard assumptions hold (that is, the solution set is nonempty and the gradient of the function is Lipschitz continuous) *and* (i) the isocost surfaces of the objective function, restricted to the solution set, are properly separated and (ii) a certain multifunction associated with the problem is locally upper Lipschitzian, then this algorithm attains a linear rate of convergence.

Key words. partially asynchronous computation, gradient projection, locally upper Lipschitzian multifunction, linear convergence

AMS(MOS) subject classifications. 49, 90

1. Introduction.

A frequently encountered problem in optimization concerns finding a stationary point of a continuously differentiable function f in $\Re^m$, the m-dimensional Euclidean space, over a closed convex set $\mathcal{X}$ in $\Re^m$. In other words, it is desired to find a solution to the fixed point problem

$$x = [x - \nabla f(x)]^+,$$

where $[\cdot]^+$ denotes the orthogonal projection onto $\mathcal{X}$, i.e., $[x]^+ = \arg\min_{y\in\mathcal{X}} ||x - y||$. In our notation, all vectors are column vectors and $||x||$ denotes the Euclidean norm of x, that is, $||x|| = \sqrt{\langle x, x\rangle}$, where $\langle x, y\rangle$ denotes the Euclidean inner product of x with y.

A well-known iterative method for solving the above problem is the gradient projection algorithm proposed by Goldstein [Gol64] and by Levitin and Polyak [LeP65]. In this algorithm, each new iterate is obtained by moving the previous iterate along the negative gradient direction, and then projecting the resulting point back onto the feasible set $\mathcal{X}$, that is,

$$x := [x - \gamma\nabla f(x)]^+, \tag{1.1}$$

where γ is some appropriately chosen positive stepsize. This algorithm possesses nice numerical properties and has been studied extensively (see [Ber76], [Ber82a], [BeG82], [CaM87], [Che84], [Dun81], [Dun87], [GaB82], [GaB84], [Gol64], [Gol74], [LeP65]).

Recently, Bertsekas and Tsitsiklis [BeT89, §7.5] (also see [Tsi84], [TBA86]) proposed a *partially asynchronous* implementation of the above algorithm, in which $\mathcal{X}$ is decomposed into the Cartesian product of closed convex sets $\mathcal{X}_1, \cdots, \mathcal{X}_n$ ($n \geq 1$) and the iteration (1.1) is distributed over n processors, with the ith processor being responsible for updating the block-component of x belonging to $\mathcal{X}_i$. Each processor carries

*Received by the editors August 17, 1990; accepted for publication (in revised form) January 11, 1991. This work was supported by U.S. Army Research Office contract DAAL03-86-K-0171 (Center for Intelligent Control Systems) and by National Science Foundation grant NSF–DDM–8903385.

†Department of Mathematics, GN-50, University of Washington, Seattle, Washington 98195.

its own estimate of the solution, communicates to the other processors by message passing, and may act independently of the other processors. Such an "asynchronous" (or "chaotic") computing environment, proposed by Chazan and Miranker [ChM69], offers several advantages over a synchronous (either sequential or parallel) computing environment: for example, the synchronization penalty is low and the fast processors need not wait for the slower ones. In addition, asynchronous computation brings forth interesting and challenging questions about the convergence of algorithms. For a detailed discussion of asynchronous computation, see [BeT89].

It is known, under a standard Lipschitz continuity condition on the gradient ∇f, that if γ is sufficiently small, then every limit point of the iterates generated by the partially asynchronous gradient projection algorithm is a stationary point [BeT89, §7.5]. However, little is known about the convergence or the rate of convergence of the iterates. In fact, even in the sequential case (i.e., the original gradient projection algorithm), fairly little is known about the rate of convergence. Rate of convergence analysis typically requires the solution points to be isolated and the objective function f to be locally strongly convex (see [LeP65], [Dun81], [Dun87]), which in general does not hold. (An exception to this is [BeG82], which proves linear rate of convergence for the case where $\mathcal{X}$ is polyhedral and f is the composition of an affine mapping with a strongly convex differentiable function.) Recently, Luo and Tseng [LuT90] (also see [LuT89], [TsL90a], [TsL90b] for related analyses) proposed a new approach to demonstrating the linear rate of convergence of iterative optimization algorithms, based on bounding the distance to the solution set from a point x near the solution set by the norm of the "residual" at x, namely,

$$x - [x - \nabla f(x)]^+.$$

Such a local "error bound" does not hold in general, but can be shown to hold for a number of important problem classes, including quadratic programs and strongly convex programs.

In this paper, we adapt the approach of Luo and Tseng to analyze the partially asynchronous gradient projection algorithm. In particular, we show that the algorithm attains a linear rate of convergence, assuming only that (i) the solution set is nonempty, (ii) f is bounded from below on $\mathcal{X}$, (iii) ∇f is Lipschitz continuous on $\mathcal{X}$, (iv) the isocost surfaces of the objective function, restricted to the solution set, are "properly separated" from each other, and (v) the above error bound holds near the solution set. Thus, even in the sequential case, our rate of convergence result appears to be a significant improvement over existing ones. (Assumptions (i) to (iv), as we shall see, hold for most problems, so the key assumption is (v).)

This paper proceeds as follows: In §2 we describe the partially asychronous gradient projection algorithm and state our main convergence result for this algorithm. In §3 we prove the main result for the special case where the computations take place sequentially. In §4, we prove the main result, building on the ideas developed in §3. In §5 we discuss possible extensions of our work.

2. Algorithm description and convergence results. We formally describe the partially asynchronous gradient projection algorithm below (also see [BeT89, §7.5]). In this algorithm, $\mathcal{X}$ is decomposed into the Cartesian product of closed convex sets $\mathcal{X}_1, \cdots \mathcal{X}_n$ $(n \geq 1)$, that is,

$$\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n. \tag{2.1}$$

According to the above product structure of $\mathcal{X}$, let the elements x of $\mathcal{X}$ be decomposed into block-components, so $x = (x_1, x_2, \cdots, x_n)$, with $x_i \in \mathcal{X}_i$. Let $\nabla_i f(x)$ denote the partial derivative of $f(x)$ with respect to x_i, and let $[x_i]_i^+$ denote the orthogonal projection of x_i onto $\mathcal{X}_i$. Then, for a given *fixed* stepsize $\gamma > 0$, the algorithm generates a sequence of iterates $\{x(1), x(2), \cdots\}$ in $\mathcal{X}$ according to the formula:

$$x_i(t+1) = \begin{cases} [x_i(t) - \gamma \nabla_i f(x^i(t))]_i^+, & \text{if } t \in T^i; \\ x_i(t), & \text{otherwise,} \end{cases} \qquad i = 1, \cdots, n, \tag{2.2}$$

where T^i is some subset of $\{0, 1, 2, \cdots\}$ and $x^i(t)$ is the vector in $\mathcal{X}$ given by

$$x^i(t) = \left(x_1(\tau_1^i(t)), \cdots, x_n(\tau_n^i(t))\right), \tag{2.3}$$

with each $\tau_j^i(t)$ some nonnegative integer not exceeding t. (The initial iterate $x(0) \in \mathcal{X}$ is assumed given.)

Roughly speaking, T^i is the set of times at which x_i is updated (by processor i); $x^i(t)$ is the solution estimate known to processor i at time t; and $\tau_j^i(t)$ is the time at which the value of x_j used by processor i at time t is generated by processor j (so $t - \tau_j^i(t)$ is effectively the communication delay from processor j to processor i at time t). A key feature of the algorithm is that the components are updated using values which may be out-of-date.

We make the standing assumption that the iterates are updated in a partially asynchronous manner.

PARTIAL ASYNCHRONISM ASSUMPTION. *There exists an integer $B \geq 1$ such that*

(a) $\{t, t+1, \cdots, t+B-1\} \cap T^i \neq \emptyset$, *for all $t \geq 0$ and all i;*

(b) $0 \leq t - \tau_j^i(t) \leq B - 1$, *for all $t \in T^i$, all j and all i.*

(Roughly speaking, the Partial Asynchronism Assumption states that no processor waits an arbitrarily long time to compute or to receive a message from another processor. The justification for this assumption is discussed in §7 of [BeT89].)

We make the following standard (and reasonable) assumptions about f and $\mathcal{X}$.

ASSUMPTION A. (a) *f is bounded from below on $\mathcal{X}$.*

(b) *The solution set $\mathcal{X}^* = \{ x \in \Re^m \mid x = [x - \nabla f(x)]^+ \}$ is nonempty.*

(c) *∇f is Lipschitz continuous on $\mathcal{X}$, that is,*

$$\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\| \quad \forall x \in \mathcal{X} \quad \forall y \in \mathcal{X}, \tag{2.4}$$

where $L > 0$ is the Lipschitz constant.

The following convergence result is due to Bertsekas and Tsitsiklis (see Proposition 5.3 in [BeT89, §7.5]).

PROPOSITION 2.1. *Under Assumption A, there exists a scalar $\gamma_0 > 0$ (depending on L, n, and B only) such that if $0 < \gamma < \gamma_0$, then any limit point of the sequence $\{x(t)\}$ generated by the partially asynchronous gradient projection algorithm* (2.2), (2.3) *is an element of $\mathcal{X}^*$.*

The above result is rather weak since it does not assert that $\{x(t)\}$ has a limit point. To prove the convergence of $\{x(t)\}$, we need to make, in addition to Assumption A, the following assumptions on f and $\mathcal{X}$.

ASSUMPTION B. (a) *There exists a scalar $\epsilon > 0$ such that*

$$x \in \mathcal{X}^*, \qquad y \in \mathcal{X}^*, \qquad f(x) \neq f(y) \Rightarrow \|x - y\| \geq \epsilon.$$

(b) *For every η there exist scalars $\delta > 0$ and $\kappa > 0$ such that*

$$\phi(x) \leq \kappa ||x - [x - \nabla f(x)]^+||, \tag{2.5}$$

for all $x \in \mathcal{X}$ with $f(x) \leq \eta$ and $||x - [x - \nabla f(x)]^+|| \leq \delta$, where we let $\phi(x) = \min_{\bar{x} \in \mathcal{X}^*} ||x - \bar{x}||$.

The main result of this paper is stated below. Its proof, which is quite involved, is given in §4.

PROPOSITION 2.2. *Under Assumptions* A *and* B, *there exists a scalar $\gamma_1 > 0$, depending on L, n, B, and $x(0)$ only, such that if $0 < \gamma < \gamma_1$, then the sequence $\{x(t)\}$ generated by the partially asynchronous gradient projection algorithm* (2.2)–(2.3) *converges at least linearly to an element of $\mathcal{X}^*$ with a B-step convergence ratio of $1 - c\gamma$, where $c > 0$ is some scalar constant.*

A few words about Assumption B are in order. Assumption B(a) is a technical assumption which states that the isocost surfaces of f, restricted to the solution set $\mathcal{X}^*$, are "properly separated" from each other. This assumption clearly holds if $\mathcal{X}^*$ is a finite set. More generally, it can be seen to hold if f takes on only a finite number of values on $\mathcal{X}^*$ or if the piecewise-smooth path connected components of $\mathcal{X}^*$ are properly separated from each other. (We say a set is *piecewise-smooth path connected* if any two points in that set can be joined by a piecewise-smooth path lying entirely in that set.) Thus, it holds automatically when f is convex (since $\mathcal{X}^*$ is then convex) or when $\mathcal{X}$ is polyhedral and f is quadratic (see Lemma 3.1 in [TsL90a]).

Assumption B(b) is closely related to the notion of a locally upper Lipschitzian multifunction (see [Rob81], [Rob82]). More precisely, for any fixed scalar η, let R be the *residual* function given by

$$R(x) = x - [x - \nabla f(x)]^+,$$

restricted to the domain $\{ x \in \mathcal{X} \mid f(x) \leq \eta \}$. Then Assumption B (b) effectively says that the inverse of R, a multifunction, is locally upper Lipschitzian at the origin or, more precisely, there exist scalars $\delta > 0$ and $\kappa > 0$ such that

$$R^{-1}(z) \subseteq R^{-1}(0) + \kappa ||z||^2 \mathcal{B},$$

for all $z \in \Re^m$ with $||z|| \leq \delta$, where $\mathcal{B}$ denotes the unit Euclidean ball in $\Re^m$.

A simple example (e.g., $\mathcal{X} = \Re$ and $f(x) = |x|^\lambda$ with $\lambda > 2$ a fixed scalar) will show that Assumption B (b) does not hold in general. On the other hand, it does hold for a number of important problem classes. For example, it holds when ∇f is strongly monotone and Lipschitz continuous (see [Pan87]). Alternatively, it holds when $\mathcal{X}$ is polyhedral and f is either quadratic (see [Rob81], [TsL90a]) or of the form

$$f(x) = g(Ex) + \langle b, x \rangle,$$

for some $k \times m$ matrix E, some $b \in \Re^m$, and some convex twice differentiable function g in $\Re^k$ with $\nabla^2 g$ positive definite everywhere (see [LuT90]). It also holds when $\mathcal{X}$ is polyhedral and f is the dual functional associated with a problem of minimizing a strictly convex function subject to linear constraints (see [TsL90b]).

3. Convergence proof for the sequential case. As the proof of Proposition 2.2 is quite intricate, it is instructive to first examine a simpler case to gain a feel for the main ideas used in the proof. In this section, we give a proof of Proposition 2.2 for

the special case of the algorithm (2.2)–(2.3) in which $B = 1$ (i.e., the original gradient projection algorithm). We remark that, even for this special case, our convergence result (see Proposition 3.1) appears to be new since it assumes neither convexity of f nor uniqueness of solution (compare with [BeG82], [BeT89, §3.5.3], [Dun81], [Dun87], [LuT90, §4], [LeP65]).

For $B = 1$, the partially asynchronous gradient projection algorithm (2.2)–(2.3) reduces to the sequential algorithm

$$x(t+1) = [x(t) - \gamma \nabla f(x(t))]^+, \qquad t = 0, 1, \cdots, \tag{3.1}$$

with $x(0) \in \mathcal{X}$ given. To analyze the convergence of $\{x(t)\}$ we need the following lemma, which follows from the observation that, for any x and d in $\Re^m$, the function $p(\gamma) = ||x - [x - \gamma d]^+||$ is monotonically increasing in $\gamma > 0$ and the function $p(\gamma)/\gamma$ is monotonically decreasing in $\gamma > 0$ (see Lemma 1 in [GaB84]; also see Lemma 2.2 in [CaM87]).

LEMMA 3.1. *For any $x \in \mathcal{X}$ and any scalar $\gamma > 0$,*

$$\min\{1, \gamma\}||x - [x - \nabla f(x)]^+|| \leq ||x - [x - \gamma \nabla f(x)]^+||.$$

We now state and prove the main result of this section. The proof is patterned after one given in §3 of [TsL90a] and is based on using the locally upper Lipschitzian condition (2.5) to show that $\{x(t)\}$ tends toward $\mathcal{X}^*$ (cf. (3.5)) and that, near $\mathcal{X}^*$, the difference in the f value of $x(t+1)$ and that of an element of $\mathcal{X}^*$ nearest to $x(t)$ is at most of the order $||x(t+1) - x(t)||^2$ (see (3.9)).

PROPOSITION 3.1. *Under Assumptions* A *and* B, *if $0 < \gamma < 2/L$, then the sequence $\{x(t)\}$ generated by the sequential gradient projection algorithm* (3.1) *converges at least linearly to an element of $\mathcal{X}^*$ with a convergence ratio of $1 - c\gamma$, where $c > 0$ is some scalar constant.*

Proof. It is well known, by using (2.4) and (3.1), that

$$f(x(t+1)) - f(x(t)) \leq -\left(\frac{1}{\gamma} - \frac{L}{2}\right)||x(t+1) - x(t)||^2 \quad \forall t. \tag{3.2}$$

(See, for example, [Gol64] or [LeP65].) Since $0 < \gamma < 2/L$ and, by Assumption A (a), f is bounded from below on $\mathcal{X}$, then (3.2) implies

$$x(t) - x(t+1) \to 0, \tag{3.3}$$

so (3.1) and Lemma 3.1 yields $x(t) - [x(t) - \nabla f(x(t))]^+ \to 0$. Since $f(x(t)) \leq f(x(0))$ for all t (cf. (3.2)), this together with Assumption B (b) implies that there exist an index $\bar{t}$ and a scalar $\kappa > 0$ (depending on $x(0)$) such that, for all $t \geq \bar{t}$, (2.5) holds with $x = x(t)$, so

$$\begin{aligned} ||x(t) - \bar{x}(t)|| &\leq \kappa ||x(t) - [x(t) - \nabla f(x(t))]^+|| \\ &\leq \kappa \max\left\{1, \frac{1}{\gamma}\right\} ||x(t) - [x(t) - \gamma \nabla f(x(t))]^+|| \\ &= \kappa \max\left\{1, \frac{1}{\gamma}\right\} ||x(t) - x(t+1)||, \end{aligned} \tag{3.4}$$

where $\bar{x}(t)$ denotes an element of $\mathcal{X}^*$ for which $||x(t) - \bar{x}(t)|| = \phi(x(t))$, the second inequality follows from Lemma 3.1, and the equality follows from (3.1). Combining (3.3) with (3.4) gives

$$x(t) - \bar{x}(t) \to 0, \tag{3.5}$$

so $\bar{x}(t) - \bar{x}(t+1) \to 0$. Then, Assumption B (a) implies that $\bar{x}(t)$ eventually settles down at some isocost surface of f, i.e., there exist an index $\hat{t} \geq \bar{t}$ and a scalar $\bar{v}$ such that

$$f(\bar{x}(t)) = \bar{v} \quad \forall t \geq \hat{t}. \tag{3.6}$$

We have, from $\bar{x}(t) \in \mathcal{X}^*$ and $x(t) \in \mathcal{X}$ that $\langle \nabla f(\bar{x}(t)), x(t) - \bar{x}(t) \rangle \geq 0$ and from the Mean Value Theorem that $f(\bar{x}(t)) - f(x(t)) = \langle \nabla f(\psi(t)), \bar{x}(t) - x(t) \rangle$, for some m-vector $\psi(t)$ lying on the line segment joining $\bar{x}(t)$ with $x(t)$. Upon summing these two relations and using (3.6), we obtain

$$\begin{aligned} \bar{v} - f(x(t)) &\leq \langle \nabla f(\psi(t)) - \nabla f(\bar{x}(t)), \bar{x}(t) - x(t) \rangle \\ &\leq ||\nabla f(\psi(t)) - \nabla f(\bar{x}(t))|| ||\bar{x}(t) - x(t)|| \\ &\leq L||\bar{x}(t) - x(t)||^2 \quad \forall t \geq \hat{t}, \end{aligned}$$

where the last inequality follows from the Lipschitz condition (2.4) and $||\psi(t) - \bar{x}(t)|| \leq ||x(t) - \bar{x}(t)||$. This, together with (3.5), yields

$$\liminf_{t \to \infty} f(x(t)) \geq \bar{v}. \tag{3.7}$$

Since $x(t+1)$ is obtained by projecting $x(t) - \gamma \nabla f(x(t))$ onto $\mathcal{X}$ (cf. (3.1)) and $\bar{x}(t) \in \mathcal{X}$, we have

$$\langle x(t) - \gamma \nabla f(x(t)) - x(t+1), x(t+1) - \bar{x}(t) \rangle \geq 0 \quad \forall t. \tag{3.8}$$

Also, by the Mean Value Theorem, for each $t \geq \hat{t}$ there exists some $\zeta(t)$ lying on the line segment joining $x(t+1)$ with $\bar{x}(t)$ such that

$$f(x(t+1)) - f(\bar{x}(t)) = \langle \nabla f(\zeta(t)), x(t+1) - \bar{x}(t) \rangle,$$

which, when combined with (3.6) and (3.8), yields

$$\begin{aligned} f(x(t+1)) - \bar{v} &= f(x(t+1)) - f(\bar{x}(t)) \\ &= \langle \nabla f(\zeta(t)), x(t+1) - \bar{x}(t) \rangle \\ &\leq \left\langle \nabla f(\zeta(t)) - \nabla f(x(t)) + \frac{1}{\gamma}(x(t) - x(t+1)), x(t+1) - \bar{x}(t) \right\rangle \\ &\leq \left(||\nabla f(\zeta(t)) - \nabla f(x(t))|| + \frac{1}{\gamma}||x(t) - x(t+1)|| \right) ||x(t+1) - \bar{x}(t)|| \\ &\leq \left(L||\zeta(t) - x(t)|| + \frac{1}{\gamma}||x(t) - x(t+1)|| \right) ||x(t+1) - \bar{x}(t)|| \\ &\leq \left(\left(L + \frac{1}{\gamma} \right) ||x(t+1) - x(t)|| + L||\bar{x}(t) - x(t)|| \right) ||x(t+1) - \bar{x}(t)|| \\ &\leq \left(\left(L + \frac{1}{\gamma} \right) ||x(t+1) - x(t)|| + L||x(t) - \bar{x}(t)|| \right) \\ &\quad \times \left(||x(t+1) - x(t)|| + ||\bar{x}(t) - x(t)|| \right) \\ &\leq \eta_1 ||x(t+1) - x(t)||^2, \end{aligned} \tag{3.9}$$

where the third inequality follows from the Lipschitz condition (2.4), the fourth inequality follows from the fact that $\zeta(t)$ lies between $x(t+1)$ and $\bar{x}(t)$, and the last

inequality follows from (3.4), with η_1 being some scalar constant depending on L, κ, and γ only.

Using (3.2) to bound the right-hand side of (3.9) gives

$$f(x(t+1)) - \bar{v} \leq \eta_2 \left(f(x(t)) - f(x(t+1))\right) \quad \forall t \geq \hat{t},$$

where η_2 is some positive scalar depending on L, κ, and γ only. Upon rearranging terms in the above relation, we obtain

$$f(x(t+1)) - \bar{v} \leq \frac{\eta_2}{1+\eta_2} \left(f(x(t)) - \bar{v}\right) \quad \forall t \geq \hat{t}.$$

On the other hand, we have from (3.7) and the fact that $f(x(t))$ is monotonically decreasing with t (cf. (3.2)), that $f(x(t)) \geq \bar{v}$ for all t, so the above relation implies that $\{f(x(t))\}$ converges at least linearly to $\bar{v}$. Since $||x(t+1) - x(t)||^2$ is of the order $f(x(t)) - f(x(t+1))$ (cf. (3.2)), this implies that $\{x(t)\}$ converges at least linearly. Since $\phi(x(t)) \to 0$ [cf. (3.5)], then the point to which $\{x(t)\}$ converges is in $\mathcal{X}^*$. That the convergence ratio is of the form $1 - \gamma c$ can be seen by explicitly writing out η_2 as a function of γ. □

We remark that we need not have assumed γ to be fixed or small in the above analysis, so long as γ is chosen so that $||x(t) - x(t+1)||^2$ is of the order $f(x(t)) - f(x(t+1))$ (cf. (3.2)). This is an important generalization since, in practice, γ is typically not fixed but determined by some linesearch rule, such as the Armijo-like rule of Bertsekas [Ber76].

4. Convergence proof for the general case. In this section we extend the analysis in §3 to prove Proposition 2.2, the main result of this paper. Our argument is very similar in idea to the proof of Proposition 3.1, but, owing to the presence of asynchronism in computations, error quantities arise in many places and have to be carefully estimated. We show that the errors caused by asynchronism are of second order in γ and are negligible when γ is small.

We assume throughout that Assumptions A and B hold. Let $\{x(t)\}$ be a sequence of iterates generated by the partially asynchronous gradient projection algorithm (2.2)–(2.3). For the moment, the only restriction that we place on the stepsize γ is that it be positive. We will, in the course of the proof, impose additional upper bounds on γ.

For each $t \geq 0$, let

$$s_i(t) = x_i(t+1) - x_i(t), \qquad i = 1, \cdots, n. \tag{4.1}$$

(For notational simplicity, we have defined $s_i(t)$ slightly differently from [BeT89, §7.5.4].) Then, by (2.2), for every i there holds

$$s_i(t) = 0, \quad \forall t \notin T^i, \tag{4.2}$$

$$s_i(t) = [x_i(t) - \gamma \nabla_i f(x^i(t))]_i^+ - x_i(t) \quad \forall t \in T^i. \tag{4.3}$$

For notational simplicity we will use $\Theta(\gamma^k)$, for any integer k, to represent any continuous function $g : (0, \infty) \to \Re$ with the property

$$\lim_{\gamma \downarrow 0} \frac{g(\gamma)}{\gamma^k} = c,$$

for some scalar $c > 0$ depending on L, n, B, and $x(0)$ only. We will implicitly assume that γ is always taken sufficiently small so that each $g(\gamma)$ encountered is positive.

First we have the following result analogous to (3.2).

LEMMA 4.1.

$$f(x(t+B)) \le f(x(t)) - \Theta(\gamma^{-1}) \sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 + \Theta(1) \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2 \quad \forall t \ge 0. \tag{4.4}$$

Proof. For any i and any $t \in T^i$, since $x_i(t) + s_i(t)$ is the orthogonal projection of $x_i(t) - \gamma \nabla_i f(x^i(t))$ onto $\mathcal{X}_i$ (cf. (4.3)) and $x_i(t) \in \mathcal{X}_i$, we have, from a well-known property of orthogonal projections, that

$$\langle s_i(t), \gamma \nabla_i f(x^i(t)) \rangle \le -||s_i(t)||^2.$$

Combining this with (4.1)–(4.2) and using (2.4) and an argument analogous to that in [BeT89, pp. 529–530] gives

$$f(x(t+1)) - f(x(t)) \le -\frac{1-\gamma L}{\gamma} ||s(t)||^2 + L \sum_{i=1}^{n} \sum_{\tau=t-B}^{t-1} ||s_i(t)|| ||s(\tau)||, \quad \forall t \ge 0.$$

By using the identity $a \cdot b \le a^2 + b^2$, we can bound the right-hand side of the above relation:

$$\begin{aligned} f(x(t+1)) - f(x(t)) &\le -\frac{1-\gamma L}{\gamma} ||s(t)||^2 + L \sum_{i=1}^{n} \sum_{\tau=t-B}^{t-1} \left(\sqrt{n} ||s_i(t)||^2 + \frac{1}{\sqrt{n}} ||s(\tau)||^2 \right) \\ &= -\frac{1-\gamma L}{\gamma} ||s(t)||^2 + L \left(B\sqrt{n} ||s(t)||^2 + \sqrt{n} \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2 \right) \\ &= -\frac{1-\gamma L - \gamma L B \sqrt{n}}{\gamma} ||s(t)||^2 + L\sqrt{n} \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2. \end{aligned}$$

Applying the above argument successively to $t, t+1, \cdots, t+B-1$ we obtain

$$\begin{aligned} f(x(t+B)) - f(x(t)) \le &- \frac{1-\gamma L - \gamma L B \sqrt{n}}{\gamma} \sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 \\ &+ LB\sqrt{n} \sum_{\tau=t-B}^{t+B-1} ||s(\tau)||^2. \end{aligned} \tag{4.5}$$

□

(We analyze the B-step decrease in f because, in the worst case, B time units can pass before any component of x is iterated upon.)

By summing (4.4) over all $t = 0, B, 2B, \cdots$, we see that, for γ sufficiently small so that the $\Theta(\gamma^{-1})$ term dominates the $\Theta(1)$ term in (4.4), there holds

$$\limsup_{t \to \infty} f(x(t)) \le f(x(0)) - \Theta(\gamma^{-1}) \sum_{\tau=0}^{\infty} ||s(\tau)||^2.$$

(In fact, it can be seen from (4.5) that it suffices to take $\gamma < L + 3LB\sqrt{n}$.) This implies that $\{f(x(t))\}$ is bounded (cf. Assumption A (a)) and

$$x(t) - x(t+1) \to 0 \tag{4.6}$$

(cf. (4.1)), so, by using (2.3)–(2.4) and (4.3) and the Partial Asynchronism Assumption, we can conclude that

$$x(t) - [x(t) - \gamma \nabla f(x(t))]^+ \to 0. \tag{4.7}$$

(See [BeT89, pp. 530–531] for a more detailed argument.) Up to this point our analysis has followed closely the proof of Proposition 5.1 in [BeT89, §7.5], but it starts to diverge from here on.

Equation (4.7) and Lemma 3.1 imply $x(t) - [x(t) - \nabla f(x(t))]^+ \to 0$, and since $\{f(x(t))\}$ is bounded, then, by Assumption B (b), there exists a threshold $\bar{t} \geq 0$ and a scalar $\kappa > 0$ (depending on $x(0)$ only) such that

$$\phi(x(t)) \leq \kappa ||x(t) - [x(t) - \nabla f(x(t))]^+|| \quad \forall t \geq \bar{t}.$$

For each t, let $\bar{x}(t)$ be an element of $\mathcal{X}^*$ satisfying $||x(t) - \bar{x}(t)|| = \phi(x(t))$. Then, we have from the above relation and Lemma 3.1, that

$$||x(t) - \bar{x}(t)|| \leq \kappa \max\left\{1, \frac{1}{\gamma}\right\} ||x(t) - [x(t) - \gamma \nabla f(x(t))]^+|| \quad \forall t \geq \bar{t}. \tag{4.8}$$

Combining (4.7) with (4.8) gives

$$x(t) - \bar{x}(t) \to 0, \tag{4.9}$$

so (4.6) yields $\bar{x}(t) - \bar{x}(t+1) \to 0$. Then, Assumption B (a) implies that $\bar{x}(t)$ eventually settles down at some isocost surface of f, so there exist an index $\hat{t} \geq \bar{t}$ and a scalar $\bar{v}$ such that

$$f(\bar{x}(t)) = \bar{v} \quad \forall t \geq \hat{t}. \tag{4.10}$$

Then, an argument identical to the proof of (3.7), with (3.5) and (3.6) replaced by (4.9) and (4.10), respectively, gives

$$\liminf_{t \to \infty} f(x(t)) \geq \bar{v}. \tag{4.11}$$

To prove our next main result (Lemma 4.4), we need the following two technical lemmas. The first lemma says that $||x(t) - [x(t) - \gamma \nabla f(x(t))]^+||^2$ is upper bounded by $\sum_{\tau=t}^{t+B-1} ||s(\tau)||^2$ plus a smaller term. The proof of this, trivial in the sequential case (cf. (3.1)), is complicated owing to the presence of asynchronism in the computations.

LEMMA 4.2. *For all $t \geq 0$, there holds*

$$||x(t) - [x(t) - \gamma \nabla f(x(t))]^+||^2 \leq \Theta(1) \sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 + \Theta(\gamma) \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2. \tag{4.12}$$

Proof. Fix any $t \in \{0, 1, \cdots\}$. For each index $i \in \{1, \cdots, n\}$, let t^i be the smallest element of T^i that exceeds t. Then (cf. (4.1), (4.2))

$$x_i(t^i) = x_i(t), \tag{4.13}$$

and, by (4.3),

$$s_i(t^i) = [x_i(t^i) - \gamma \nabla_i f(x^i(t^i))]_i^+ - x_i(t^i). \tag{4.14}$$

Also, by part (a) of the Partial Asynchronism Assumption, there holds $t \le t^i \le t + B - 1$. Combining (4.13) with (4.14), we have

$$\begin{aligned} ||s_i(t^i)|| &= ||[x_i(t) - \gamma \nabla_i f(x^i(t^i))]_i^+ - x_i(t)|| \\ &\ge ||[x_i(t) - \gamma \nabla_i f(x(t))]_i^+ - x_i(t)|| - \gamma ||\nabla_i f(x(t)) - \nabla_i f(x^i(t^i))|| \\ &\ge ||[x_i(t) - \gamma \nabla_i f(x(t))]_i^+ - x_i(t)|| - \gamma L ||x(t) - x^i(t^i)||, \end{aligned}$$

where the last inequality follows from the Lipschitz condition (2.4). By using the identity $(a - \gamma b)^2 \ge (1 - \gamma)a^2 - \gamma b^2$, we obtain from the above relation that

$$||s_i(t^i)||^2 \ge (1 - \gamma) ||[x_i(t) - \gamma \nabla_i f(x(t))]_i^+ - x_i(t)||^2 - \gamma L^2 ||x(t) - x^i(t^i)||^2. \tag{4.15}$$

Also, since $t \le t^i \le t + B - 1$ so that (by part (b) of the Partial Asynchronism Assumption) $t - B + 1 \le \tau_j^i(t^i) \le t + B - 1$, we have from (4.1) that, for all j,

$$\begin{aligned} ||x_j(t) - x_j(\tau_j^i(t^i))||^2 &\le \left(\sum_{\tau=t-B+1}^{t+B-1} ||s_j(\tau)|| \right)^2 \\ &\le 2B \sum_{\tau=t-B+1}^{t+B-1} ||s_j(\tau)||^2, \end{aligned}$$

where the second inequality follows from the identity $(a_1 + \cdots + a_{2B})^2 \le 2B(a_1)^2 + \cdots + 2B(a_{2B})^2$. Summing the above relation over all j and using (2.3) yields

$$||x(t) - x^i(t^i)||^2 \le 2B \sum_{\tau=t-B}^{t+B-1} ||s(\tau)||^2.$$

Using the above to bound the right-hand side of (4.15) then gives

$$||s_i(t^i)||^2 \ge (1 - \gamma) ||[x_i(t) - \gamma \nabla_i f(x(t))]_i^+ - x_i(t)||^2 - 2\gamma L^2 B \sum_{\tau=t-B}^{t+B-1} ||s(\tau)||^2.$$

Since the choice of i was arbitrary, the above relation holds for all $i \in \{1, \cdots, n\}$, which when summed over all i and using the "obvious" inequality [cf. $t \le t^i \le t+B-1$]

$$\sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 = \sum_{i=1}^{n} \sum_{\tau=t}^{t+B-1} ||s_i(\tau)||^2 \ge \sum_{i=1}^{n} ||s_i(t^i)||^2,$$

then gives

$$\sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 \ge (1 - \gamma) ||[x(t) - \gamma \nabla f(x(t))]^+ - x(t)||^2 - 2\gamma L^2 B n \sum_{\tau=t-B}^{t+B-1} ||s(\tau)||^2.$$

Taking $\gamma < 1$ and rearranging terms in the above relation proves (4.12). □

We next have a technical lemma on the behaviour of f over $\mathcal{X}$. Its proof is given in §6.

LEMMA 4.3. *For any x and $x^1, \cdots, x^n$ in $\Re^m$ and any $\bar{x} \in \mathcal{X}$, there holds*

(4.16)
$$f(z) - f(\bar{x}) \leq \Theta(\gamma^{-2})||x - [x - \gamma\nabla f(x)]^+||^2 + \Theta(1)\left(||x - \bar{x}||^2 + \sum_{i=1}^{n} ||x - x^i||^2\right),$$

where z is the m-vector with components $z_i = [x_i - \gamma\nabla_i f(x^i)]_i^+$, $i = 1, \cdots, m$.

By using (4.8) and (4.10) together with Lemmas 4.2 and 4.3, we can now upper bound $f(x(t+B)) - \bar{v}$ in a manner analogous to (3.9).

LEMMA 4.4. *For all $t \geq \hat{t}$, there holds*

$$f(x(t+B)) - \bar{v} \leq \Theta(\gamma^{-2}) \sum_{\tau=t}^{t+B-1} ||s(\tau)||^2 + \Theta(\gamma^{-1}) \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2. \tag{4.17}$$

Proof. Fix any $t \geq \hat{t}$. For each i, let t^i denote the smallest element of T^i exceeding t. Then, by (2.2),

$$x_i(t^i + 1) = [x_i(t) - \gamma\nabla_i f(x^i(t^i))]_i^+ \quad \forall i, \tag{4.18}$$

and, by part (a) of the Partial Asynchronism Assumption,

$$t \leq t^i \leq t + B - 1 \quad \forall i. \tag{4.19}$$

Let us apply Lemma 4.3 with $x = x(t)$, $x^i = x^i(t^i)$ for all i, and $\bar{x} = \bar{x}(t)$. This then gives

$$f(z) - f(\bar{x}(t)) \leq \Theta(\gamma^{-2})r(t) + \Theta(1)\left(||x(t) - \bar{x}(t)||^2 + \sum_{i=1}^{n} ||x(t) - x^i(t^i)||^2\right),$$

where z is the m-vector whose ith component z_i is $x_i(t^i + 1)$ [cf. (4.18)] and for convenience we have let $r(t) = ||x(t) - [x(t) - \gamma\nabla f(x(t))]^+||^2$. By applying (4.8) and (4.10) to the above relation, we obtain the simpler bound

$$\begin{aligned} f(z) - \bar{v} &\leq \Theta(\gamma^{-2})r(t) + \Theta(1)\sum_{i=1}^{n} ||x(t) - x^i(t^i)||^2 \\ &= \Theta(\gamma^{-2})r(t) + \Theta(1)\sum_{i=1}^{n}\sum_{j=1}^{n} ||x_j(t) - x_j(\tau_j^i(t^i))||^2, \end{aligned}$$

where the equality follows from (2.3). Since (4.19) holds, then part (b) of the Partial Asynchronism Assumption implies $t - B + 1 \leq \tau_j^i(t^i) \leq t + B - 1$ for all i and all j, so the above relation, together with (4.1), yields

$$\begin{aligned} f(z) - \bar{v} &\leq \Theta(\gamma^{-2})r(t) + \Theta(1)\sum_{j=1}^{n}\sum_{\tau=t-B+1}^{t+B-1} ||s_j(\tau)||^2 \\ &= \Theta(\gamma^{-2})r(t) + \Theta(1)\sum_{\tau=t-B+1}^{t+B-1} ||s(\tau)||^2. \end{aligned} \tag{4.20}$$

Also, we have from (4.1), (4.19), and the definition of z_i that

$$x_i(t+B) - z_i = x_i(t+B) - x_i(t^i+1) = \sum_{\tau=t^i+1}^{t+B-1} s_i(\tau) \quad \forall i,$$

so an argument similar to the proof of (4.4) yields

$$f(x(t+B)) \le f(z) + \Theta(1) \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2.$$

This, combined with (4.20), and then using the definition of $r(t)$ and (4.12), gives (4.17). □

By using the bounds (4.4), (4.11), and (4.17), we can now prove the linear convergence of $\{x(t)\}$. To simplify the notation, let

$$\alpha(t) = f(x(t)) - \bar{v},$$
$$\beta(t) = \sum_{\tau=t-B}^{t-1} ||s(\tau)||^2,$$

for all $t \ge \hat{t}$. Then, we have from (4.4), (4.11), and (4.17), respectively, that, for any $t \ge \hat{t}$,

$$\alpha(t+B) \le \alpha(t) - \gamma^{-1} A_1 \beta(t+B) + A_2 \beta(t), \tag{4.21}$$

$$0 \le \liminf_{\tau \to \infty} \alpha(\tau), \tag{4.22}$$

$$\alpha(t+B) \le \gamma^{-2} A_3 \beta(t+B) + \gamma^{-1} A_3 \beta(t), \tag{4.23}$$

where A_1, A_2, A_3 are positive scalars depending on L, n, B, and $x(0)$ only. (Of course, we always assume, implicitly, that γ is sufficiently small.) Note that we are now explicitly writing out the constant in the $\Theta(\cdot)$ notation. For this part of the proof, the constant matters. Our goal will be to show, by induction, that $\{\alpha(t)\}$ and, in particular, $\{\beta(t)\}$ converge at least linearly (see Lemma 4.5). (Note that $\beta(t) \ge 0$ but, in contrast to the sequential case, $\alpha(t)$ may be negative. This fortunately does not complicate our proof to any significant degree.)

Fix any $t \ge \hat{t} + B$. Applying (4.23) to bound the $\beta(t+B)$ term in (4.21) and then rearranging terms gives

$$(1 + \gamma A_1/A_3)\alpha(t+B) \le \alpha(t) + (A_1 + A_2)\beta(t).$$

Also, by substituting $t-B$ for t in (4.21) and rearranging terms, we obtain $\gamma^{-1} A_1 \beta(t) \le \alpha(t-B) - \alpha(t) + A_2 \beta(t-B)$, which, when applied to bound the right-hand side of the above relation, yields

$$(1 + \gamma A_1/A_3)\alpha(t+B) \le \alpha(t) + \gamma(1 + A_2/A_1)\left(\alpha(t-B) - \alpha(t) + A_2\beta(t-B)\right).$$

After rearranging terms, we obtain

$$\alpha(t+B) \le \frac{1}{1 + \gamma A_1/A_3}\left((1-\gamma A_4)\alpha(t) + \gamma A_4 \left(\alpha(t-B) + A_2\beta(t-B)\right)\right), \tag{4.24}$$

where for convenience we let $A_4 = 1 + A_2/A_1$. Also, for any integer $k \geq 2$, we have from repeated applications of (4.21) that

$$\alpha(t+kB) \leq \alpha(t) - \gamma^{-1}A_1 \sum_{l=1}^{k} \beta(t+lB) + A_2 \sum_{l=0}^{k-1} \beta(t+lB)$$
$$= \alpha(t) - (\gamma^{-1}A_1 - A_2) \sum_{l=1}^{k-1} \beta(t+lB) - \gamma^{-1}A_1\beta(t+kB) + A_2\beta(t).$$

By taking $\gamma < A_1/A_2$, we then obtain from the nonnegativity of $\beta(\tau)$, for all τ, that

$$\alpha(t+kB) \leq \alpha(t) - (\gamma^{-1}A_1 - A_2)\beta(t+B) + A_2\beta(t).$$

By letting $k \to \infty$ in the above relation, we obtain from (4.22) that

$$0 \leq \alpha(t) - (\gamma^{-1}A_1 - A_2)\beta(t+B) + A_2\beta(t),$$

which, upon rearranging terms, gives

$$\beta(t+B) \leq \frac{\gamma}{A_1 - \gamma A_2}(\alpha(t) + A_2\beta(t)). \tag{4.25}$$

Fix any two scalars $a > 0$ and $b > 0$ satisfying

$$8A_3A_4A_2b = A_1a, \tag{4.26}$$

with a and b taken sufficiently large so that

$$\alpha(\hat{t}) \leq a, \qquad \alpha(\hat{t}+B) \leq a, \qquad \beta(\hat{t}) \leq b, \qquad \beta(\hat{t}+B) \leq b. \tag{4.27}$$

Also let

$$c = \frac{A_1}{2A_3 + 2A_1}. \tag{4.28}$$

By using (4.24)–(4.28), we obtain the following main result of this section.

LEMMA 4.5. *There exists a scalar $\gamma_1 > 0$ (depending on L, n, B, and $x(0)$ only) such that if $0 < \gamma < \gamma_1$, then, for all $r = 0, 1, 2, \cdots$, there holds*

$$\alpha(\hat{t}+rB) \leq a\rho^{r-1}, \tag{4.29}$$
$$\beta(\hat{t}+rB) \leq b\rho^{r-1}, \tag{4.30}$$

where

$$\rho = 1 - \gamma c. \tag{4.31}$$

Proof. Suppose that γ is sufficiently small so that furthermore

$$\gamma < 1/A_4, \tag{4.32a}$$
$$\gamma \leq 1/(2c), \tag{4.32b}$$
$$\gamma \leq A_1/(8A_3A_4c), \tag{4.32c}$$
$$\gamma < 1, \tag{4.32d}$$
$$\gamma(a/b + A_2) \leq (A_1 - \gamma A_2)(1 - \gamma c). \tag{4.32e}$$

We will show by induction on r that (4.29) and (4.30) hold for all $r \geq 0$.

By (4.27), both (4.29) and (4.30) hold for $r = 0, 1$. Suppose that (4.29) and (4.30) hold for all r from 0 up to some $k \geq 1$. We show below that (4.29) and (4.30) hold for $r = k + 1$, which would then complete the induction on r and show that (4.29) and (4.30) hold for all $r \geq 0$. For convenience, we denote $t = \hat{t} + kB$ in what follows.

First we show that

$$\alpha(t + B) \leq a\rho^k. \tag{4.33}$$

Since (4.29) and (4.30) hold for all r up to k, we obtain from (4.24) and (4.32a) that

$$\begin{aligned}
\alpha(t+B) &\leq \frac{1}{1+\gamma A_1/A_3}\left((1-\gamma A_4)a\rho^{k-1} + \gamma A_4\left(a\rho^{k-2} + A_2 b\rho^{k-2}\right)\right) \\
&\leq \frac{1}{1+\gamma A_1/A_3}\left(1 - \gamma A_4 + \gamma A_4(1 + A_2 b/a)(1 + 2\gamma c)\right) a\rho^{k-1} \\
&= \frac{1}{1+\gamma A_1/A_3}\left(1 + \gamma^2 2A_4 c + \gamma A_4(1+2\gamma c)A_2 b/a\right) a\rho^{k-1} \\
&\leq \frac{1}{1+\gamma A_1/A_3}\left(1 + \gamma A_1/(2A_3)\right) a\rho^{k-1} \\
&= \left(1 - \frac{\gamma A_1}{2A_3 + \gamma 2A_1}\right) a\rho^{k-1} \\
&\leq \left(1 - \frac{\gamma A_1}{2A_3 + 2A_1}\right) a\rho^{k-1},
\end{aligned}$$

where the second inequality follows from the bound $\rho^{-1} \leq 1 + 2c\gamma$ [cf. (4.31), (4.32b)], the third inequality follows from (4.26) and (4.32b)–(4.32c), and the last inequality follows from (4.32d). The above relation, together with (4.28) and (4.31), proves (4.33).

Next we show that

$$\beta(t + B) \leq b\rho^k. \tag{4.34}$$

Since (4.29) and (4.30) hold for all r from 0 up to k, we have from (4.25) that

$$\begin{aligned}
\beta(t+B) &\leq \frac{\gamma}{A_1 - \gamma A_2}(a\rho^{k-1} + A_2 b\rho^{k-1}) \\
&= \frac{\gamma(a/b + A_2)}{A_1 - \gamma A_2} b\rho^{k-1}.
\end{aligned}$$

This, combined with (4.32e) and (4.31) shows that (4.34) holds.

Since (4.33) and (4.34) hold, then (4.29) and (4.30) hold for $r = k + 1$. □

Lemma 4.5 implies that $\{\beta(t)\}$ converges at least linearly with a B-step convergence ratio of $1 - \gamma c$. Since $||x(t) - x(t - B)||^2 \leq B\beta(t)$ for all t (cf. (4.1) and the definition of $\beta(t)$), this shows that $\{x(t)\}$ converges at least linearly with a B-step convergence ratio of $\sqrt{1 - \gamma c}$, which is at most $1 - \gamma c/2$. Since $\phi(x(t)) \to 0$ (cf. (4.9)), it follows that the point to which $\{x(t)\}$ converges is an element of $\mathcal{X}^*$.

5. Extensions. For simplicity, we have assumed that ∇f is Lipschitz continuous everywhere on $\mathcal{X}$ (cf. (2.4)), but this need not be so. More generally, it suffices that f tends to ∞ at any boundary point of its effective domain and that ∇f is Lipschitz continuous on each level set of f, intersected with $\mathcal{X}$.

In [TsB86] (also see §7.6 of [BeT89]), a distributed asynchronous routing algorithm is discussed. This algorithm is based on the idea of gradient projection and, by making suitable modifications to our analysis, it is possible to show that, under conditions analogous to Assumptions A and B, this algorithm also attains a linear rate of convergence.

A drawback of our main result (Proposition 2.2) is that convergence requires the algorithm to take very small steps. Intuitively, if f is approximately separable with respect to the components x_i (that is, $f(x) \approx \sum_i f_i(x_i)$ for some functions f_i), then the algorithm should be able to take much larger steps. This notion can be made precise by incorporating the effect of second-order quantities such as $\partial f/\partial x_i \partial x_j$ (assuming that f is twice differentiable) into the convergence analysis.

6. Proof of Lemma 4.3. For each i, since z_i is the orthogonal projection of $x_i - \gamma \nabla_i f(x^i)$ onto the closed convex set $\mathcal{X}_i$ and (cf. $\bar{x} \in \mathcal{X}$ and (2.1)) $\bar{x}_i \in \mathcal{X}_i$, we have

$$\langle z_i - \bar{x}_i, x_i - \gamma \nabla_i f(x^i) - z_i \rangle \geq 0.$$

Also, by the Mean Value Theorem, there exists some ζ lying on the line segment joining z with $\bar{x}$ such that $f(z) - f(\bar{x}) = \langle z - \bar{x}, \nabla f(\zeta) \rangle$, so the above relation yields

$$\begin{aligned} f(z) - f(\bar{x}) &= \langle z - \bar{x}, \nabla f(\zeta) \rangle \\ &\leq \sum_{i=1}^{n} \left\langle z_i - \bar{x}_i, \nabla_i f(\zeta) - \nabla_i f(x^i) + \frac{1}{\gamma}(x_i - z_i) \right\rangle \\ &\leq \sum_{i=1}^{n} ||z_i - \bar{x}_i|| \left(||\nabla_i f(\zeta) - \nabla_i f(x^i)|| + \frac{1}{\gamma} ||x_i - z_i|| \right) \\ &\leq \sum_{i=1}^{n} ||z_i - \bar{x}_i|| \left(L||\zeta - x^i|| + \frac{1}{\gamma} ||x_i - z_i|| \right) \\ &\leq \sum_{i=1}^{n} ||z - \bar{x}|| \left(L||\zeta - x^i|| + \frac{1}{\gamma} ||x - z|| \right), \end{aligned} \tag{6.1}$$

where the third inequality follows from the Lipschitz condition (2.4). Now we bound the right-hand side of (6.1). Since ζ is between z and $\bar{x}$, we have $||\zeta - x|| \leq ||z - x|| + ||\bar{x} - x||$, so that

$$\begin{aligned} ||\zeta - x^i|| &\leq ||\zeta - x|| + ||x - x^i|| \\ &\leq ||z - x|| + ||\bar{x} - x|| + ||x - x^i||. \end{aligned} \tag{6.2}$$

Let $\bar{z} = [x - \gamma \nabla f(x)]^+$. Then we have, from the definition of z and the nonexpansive property of the projection operators $[\cdot]_i^+$, $i = 1, \cdots, n$, that

$$\begin{aligned} ||z - \bar{z}||^2 &= \sum_{i=1}^{n} ||z_i - \bar{z}_i||^2 \\ &= \sum_{i=1}^{n} ||[x_i - \gamma \nabla_i f(x^i)]_i^+ - [x_i - \gamma \nabla_i f(x)]_i^+||^2 \\ &\leq \sum_{i=1}^{n} \gamma^2 ||\nabla_i f(x^i) - \nabla_i f(x)||^2 \\ &\leq \sum_{i=1}^{n} \gamma^2 L^2 ||x^i - x||^2, \end{aligned} \tag{6.3}$$

where the last inquality follows from the Lipschitz condition (2.4).

By bounding the right-hand side of (6.1) using (6.2) and the triangle inequality, we have

$$\begin{aligned} f(z) - f(\bar{x}) &\leq \sum_{i=1}^{n} ||z - \bar{x}|| \left(L||\bar{x} - x|| + L||x - x^i|| + (L + \frac{1}{\gamma})||z - x|| \right) \\ &\leq \sum_{i=1}^{n} (||z - \bar{z}|| + ||\bar{z} - x|| + ||x - \bar{x}||) \Bigg(L||\bar{x} - x|| + L||x - x^i|| \\ &\qquad + \left(L + \frac{1}{\gamma} \right) (||z - \bar{z}|| + ||\bar{z} - x||) \Bigg) \\ &\leq n \left(3 \left(L + \frac{1}{\gamma} \right)^2 + 4 \right) (||z - \bar{z}||^2 + ||\bar{z} - x||^2) + n(3L^2 + 4)||x - \bar{x}||^2 \\ &\quad + 3L^2 \sum_{i=1}^{n} ||x - x^i||^2, \end{aligned}$$

where the last inequality follows from expanding out the product in the line above and then using the bound $a \cdot b \leq a^2 + b^2$ on each term of the expansion. Using (6.3) to bound the $||z - \bar{z}||^2$ term in the above relation, obtain (4.16).

Acknowledgment. Thanks are due to Z.-Q. Luo for his many helpful comments on an earlier draft of this paper.

REFERENCES

[Ber76] D. P. BERTSEKAS, *On the Goldstein–Levitin–Polyak gradient projection method*, IEEE Trans. Automat. Control, AC-21 (1976), pp. 174–184.

[Ber82a] ———, *Projected Newton Methods for Optimization Problems with Simple Constraints*, SIAM J. Control. Optim., 20 (1982), pp. 221–246.

[Ber82b] ———, *Constrained Optimization and Lagrange Multiplier Methods*, Academic Press, New York, NY, 1982.

[BeG82] D. P. BERTSEKAS AND E. M. GAFNI, *Projection methods for variational inequalities with application to the traffic assignment problem*, Math. Programming Stud., 17 (1982), pp. 139–159.

[BeT89] D. P. BERTSEKAS AND J. N. TSITSIKLIS, *Parallel and Distributed Computation: Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1989.

[CaM87] P. H. CALAMAI AND J. J. MORÉ, *Projected gradient methods for linearly constrained problems*, Math. Programming, 39 (1987), pp. 93–116.

[ChM69] D. CHAZAN AND W. L. MIRANKER, *Chaotic relaxation*, Linear Algebra Appl., 2 (1969), pp. 199–222.

[Che84] Y. C. CHENG, *On the gradient–projection method for solving the nonsymmetric linear complementarity problem*, J. Optim. Theory Appl., 43 (1984), pp. 527–541.

[Dun81] J. C. DUNN, *Global and asymptotic convergence rate estimates for a class of projected gradient processes*, SIAM J. Control Optim., 19 (1981), pp. 368–400.

[Dun87] ———, *On the convergence of projected gradient processes to singular critical points*, J. Optim. Theory Appl., 55 (1987), pp. 203–216.

[GaB82] E. M. GAFNI AND D. P. BERTSEKAS, *Convergence of a Gradient Projection Method*, Report No. P–1201, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 1982.

[GaB84] E. M. GAFNI AND D. P. BERTSEKAS, *Two–metric projection methods for constrained optimization*, SIAM J. Control Optim., 22 (1984), pp. 936–964.

[Gol64] A. A. GOLDSTEIN, *Convex programming in Hilbert space*, Bull. Amer. Math. Soc., 70 (1964), pp. 709–710.

[Gol74] A. A. GOLDSTEIN, *On gradient projection*, in Proc. 12th Annual Allerton Conference on Circuits and Systems, Allerton Park, IL, 1974, pp. 38–40.

[LeP65] E. S. LEVITIN AND B. T. POLYAK, *Constrained minimization methods*, Zh. Vychisl. Mat. i Mat. Fiz., 6 (1965), pp. 787-823; English translation in USSR Comput. Math. Phys., 6 (1965), pp. 1–50.

[LuT89] Z.-Q. LUO AND P. TSENG, *On the convergence of the coordinate descent method for convex differentiable minimization*, Report No. P–1924, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, December 1989 (revised July 1990); J. Optim. Theory Appl., 72 (1992), to appear.

[LuT90] ———, *On the linear convergence of descent methods for convex essentially smooth minimization*, Report No. P–1979, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, June 1990; SIAM J. Control Optim., to appear.

[Mor89] J. J. MORÉ, *Gradient projection techniques for large-scale optimization problems*, in Proc. 28th Conference on Decision and Control, Tampa, FL, December 1989.

[Pan87] J.-S. PANG, *A posteriori error bounds for the linearly-constrained variational inequality problem*, Math. Oper. Res., 12 (1987), pp. 474–484.

[Rob81] S. M. ROBINSON, *Some continuity properties of polyhedral multifunctions*, Math. Programming Stud., 14 (1981), pp. 206–214.

[Rob82] ———, *Generalized equations and their solutions, part* II: *Applications to nonlinear programming*, Math. Programming Stud., 19 (1982), pp. 200–221.

[TsL90a] P. TSENG AND Z.-Q. LUO, *Error bound and convergence analysis of matrix splitting algorithms for the affine variational inequality problem*, Report P-1988, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, June 1990; SIAM J. Optimization, to appear.

[TsL90b] ———, *On the linear convergence of dual ascent methods for minimizing a strictly convex function subject to linear constraints*, in preparation.

[Tsi84] J. N. TSITSIKLIS, *Problems in decentralized decision making and computation*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1984.

[TsB86] J. N. TSITSIKLIS AND D. P. BERTSEKAS, *Distributed asynchronous optimal routing in data networks*, IEEE Trans. Automat. Control, AC-31 (1986), pp. 325–332.

[TBA86] J. N. TSITSIKLIS, D. P. BERTSEKAS, AND M. ATHANS, *Distributed asynchronous deterministic and stochastic gradient optimization algorithms*, IEEE Trans. Automat. Control, AC-31 (1986), pp. 803–812.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 620–642, November 1991

012

PARTITIONED DYNAMIC PROGRAMMING FOR OPTIMAL CONTROL*

STEPHEN J. WRIGHT†

Abstract. Parallel algorithms for the solution of linear-quadratic optimal control problems are described. The algorithms are based on a straightforward decomposition of the domain of the problem, and are related to multiple shooting methods for two-point boundary value problems. Their arithmetic cost is approximately twice that of the serial dynamic programming approach; however, they have the advantage that they can be efficiently implemented on a wide variety of parallel architectures. Extension to the case in which there are box constraints on the controls is simple. The algorithms can be used to solve linear-quadratic subproblems arising from the application of Newton's method or two-metric gradient projection methods to nonlinear problems.

Key words. discrete-time optimal control, parallel algorithms, dynamic programming, multiple shooting

AMS(MOS) subject classifications. 49M40, 65Y05, 90C39

1. Introduction.

The unconstrained N-stage discrete-time optimal control problem with Bolza objectives has the form

$$\min_u F(u) \stackrel{\text{def}}{=} \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i, u_i) \tag{1}$$

$$x_{i+1} = f_i(x_i, u_i), \qquad i = 1, \cdots, N, \qquad x_1 = a \text{ (fixed)}; \tag{2}$$

where $x_i \in R^n$, $i = 1, \cdots, N+1$, and $u_i \in R^m$, $i = 1, \cdots, N$. The x_i variables are usually referred to as *states* and the u_i as *controls*. (The *costates* p_i are the Lagrange multipliers corresponding to the constraints (2).) In Dunn and Bertsekas [5] and Wright [14], algorithms which exhibit local quadratic convergence to nondegenerate minimizers of (1)–(2) are discussed. These algorithms are Newton's method (Algorithm III of [14]), and variants of sequential quadratic programming (Algorithms I and II of [14]), and they all require the solution of a linear-quadratic subproblem of the following form at each iteration:

$$\begin{aligned} \min_{v,y} \ & \textstyle\sum_{i=1}^{N} r_i^T v_i + z_i^T y_i + \frac{1}{2}(y_i^T Q_i y_i + 2 y_i^T R_i v_i + v_i^T S_i v_i) \\ & + z_{N+1}^T y_{N+1} + \tfrac{1}{2} y_{N+1}^T Q_{N+1} y_{N+1}, \\ & y_{i+1} = A_i y_i + B_i v_i + s_i, \qquad i = 1, \cdots, N, \qquad y_1 = s_0. \end{aligned} \tag{3}$$

Here, y_i, v_i, q_i denote the steps in x_i, u_i, p_i, respectively,

$$\begin{aligned} & Q_i = \frac{\partial^2 \mathcal{L}}{\partial x_i^2}; \qquad R_i = \frac{\partial^2 \mathcal{L}}{\partial x_i \partial u_i}; \qquad S_i = \frac{\partial^2 \mathcal{L}}{\partial u_i^2}; \\ & A_i = \frac{\partial f^i}{\partial x_i}; \qquad B_i = \frac{\partial f^i}{\partial u_i}; \qquad z_i = \frac{\partial \mathcal{L}}{\partial x_i}; \qquad r_i = \frac{\partial \mathcal{L}}{\partial u_i}; \\ & s_i = -x_{i+1} + f_i(x_i, u_i), \end{aligned} \tag{4}$$

* Received by the editors August 21, 1990; accepted for publication (in revised form) March 1, 1991. This research was supported by the Applied Mathematical Sciences subprogram of the Office of Energy research, United States Department of Energy, contract W-31-109-Eng-38, and by the National Science Foundation under contract DMS-8900984.

† Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439.

and $\mathcal{L}$ is the Lagrangian for (1)–(2), specifically:

$$\mathcal{L}(u,x,p) = \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i,u_i) - \sum_{i=1}^{N} p_{i+1}^T(x_{i+1} - f_i(x_i,u_i)). \tag{5}$$

The first-order conditions for a set of vectors y_i, v_i, q_i to be a solution of (3) are the following system of linear equations:

$$r_i + R_i^T y_i + S_i v_i + B_i^T q_{i+1} = 0, \qquad i = 1,\cdots,N, \tag{6}$$

$$z_i + R_i v_i + Q_i y_i - q_i + A_i^T q_{i+1} = 0, \qquad i = 1,\cdots,N, \tag{7}$$

$$z_{N+1} + Q_{N+1} y_{N+1} - q_{N+1} = 0, \tag{8}$$

$$s_i - y_{i+1} + A_i y_i + B_i v_i = 0, \qquad i = 1,\cdots,N. \tag{9}$$

When the initial value $y_1 = s_0$ is included, the unknowns are ordered as

$$(q_1, y_1, v_1, q_2, y_2, v_2, \cdots, q_{N+1}, y_{N+1}),$$

and the equations are ordered appropriately, the coefficient matrix for this linear system becomes

$$\begin{bmatrix}
0 & -I & & & & & & & & & \\
-I & Q_1 & R_1 & A_1^T & & & & & & & \\
 & R_1^T & S_1 & B_1^T & & & & & & & \\
 & A_1 & B_1 & 0 & -I & & & & & & \\
 & & & -I & Q_2 & R_2 & A_2^T & & & & \\
 & & & & R_2^T & S_2 & B_2^T & & & & \\
 & & & & A_2 & B_2 & 0 & -I & & & \\
 & & & & & & -I & Q_3 & R_3 & A_3^T & \\
 & & & & & & & R_3^T & S_3 & B_3^T & \\
 & & & & & & & A_3 & B_3 & 0 & -I \\
 & & & & & & & & & -I & Q_4 \\
 & & & & & & & & & & \ddots \quad \ddots
\end{bmatrix} \tag{10}$$

with a right-hand side of

$$(-s_0^T, -z_1^T, -r_1^T, -s_1^T, -z_2^T, -r_2^T, -s_2^T, \cdots, -s_N^T, -z_{N+1}^T)^T. \tag{11}$$

The matrix is symmetric, with dimension $2n(N+1) + mN$, and a half-bandwidth of $(2n + m - 1)$.

A necessary and sufficient condition for (3) to have a unique, finite solution is that

$$\sum_{i=1}^{N} \frac{1}{2}(y_i^T Q_i y_i + 2 y_i^T R_i v_i + v_i^T S_i v_i) + \frac{1}{2} y_{N+1}^T Q_{N+1} y_{N+1} > 0 \tag{12}$$

for all y_i, v_i which satisfy

$$y_{i+1} = A_i y_i + B_i v_i, \qquad i = 1,\cdots,N, \qquad y_1 = 0. \tag{13}$$

Under this assumption, the point which satisfies the first-order conditions (6)–(9) is identical to the solution of (3). Second-order sufficient conditions for the vectors x_i^*,

u_i^*, p_i^* to be a solution of (1)–(2) include the requirement that (12)–(13) be satisfied when Q_i, R_i, etc., are defined by (4), evaluated at x_i^*, u_i^*, p_i^*. When the functions ℓ_i and f_i are sufficiently smooth, then (12)–(13) will also hold in some neighborhood of this optimal point. If (12)–(13) fails to hold, then the solution of (6)–(9) may be a saddle point for (3), and hence may not be a suitable search direction for the nonlinear algorithm. Clearly, modifications to the basic algorithms of [5] and [14] are needed to ensure that satisfactory global convergence properties will hold. Dunn and Bertsekas [5] suggest a Levenberg–Marquardt strategy, in which damping terms are added to the matrices S_i of (10). This does not destroy the structure of the matrix (10).

Another possible difficulty arises when the second derivatives of the functions f_i and ℓ_i are difficult to obtain. In this case, "pointwise" quasi-Newton methods, or finite differencing, can be used to replace the matrices Q_i, R_i, S_i by suitable approximations. (See Kelley and Sachs [8] for a pointwise quasi-Newton method for the infinite-dimensional analog of (1)–(2).) Again, the linear algebra task remains the same.

When pointwise constraints are applied to the controls u_i (for example, bounds on their magnitudes), projected gradient or active set methods give rise to linear systems similar to (10)–(11), in which transformation of the variable space leads to "reduction" of S_i, R_i, B_i, r_i. This is discussed further in §4.

From the above discussion, it should be clear that, implicitly or explicitly, the task of solving a problem of the form of (3) (or, alternatively, (10)–(11)) is at the heart of most practical algorithms for solving (1)–(2). Indeed, computationally speaking, it is usually the most time-consuming task. In this paper, we focus on this "inner loop," and present methods for efficiently solving (10)–(11) on multiprocessor computers. These methods take more account of the structure of the problem than the approach of [14], in which a general parallel bandsolver, with some modifications, was applied directly to (10)–(11). They are therefore faster, and lend themselves better to recursive, or "multilevel," implementation (see §3.3). The tradeoff is that, in contrast to the bandsolve approach, they may fail to find a solution to the system (10)–(11), even when the coefficient matrix is nonsingular. Such an occurrence is anticipated to be rare (it has not yet been observed on any test problems), but it would be wise to retain the bandsolve approach as a backup strategy.

The remainder of the paper proceeds as follows. In §2, known serial algorithms for efficiently solving (6)–(9) are described. Partitioning of the problem by what amounts to a domain decomposition, and corresponding modifications of the algorithms, are described in §3. In §4 we show how to modify the algorithms when constraints on the controls u_i are present, while in §5 some timing analysis of the performance of recursive implementations of these algorithms on multiprocessor architectures is given. Some results from an implementation on a shared-memory architecture are presented in §6. Finally, in §7, we discuss the continuous-time analog of (3), namely, the problem of finding functions $y : [0,T] \to R^n$ and $v : [0,T] \to R^m$ such that

$$\begin{aligned} (14)\quad & \int_0^T \tfrac{1}{2}y(t)^T Q(t)y(t) + y(t)^T R(t)v(t) + \tfrac{1}{2}v(t)^T S(t)v(t) + v(t)^T r(t) + y(t)^T z(t)\, dt \\ & \qquad + \tfrac{1}{2}y(T)^T Q_f Y(T) + y(T)^T z_f, \\ & \text{where} \quad \dot{y} = A(t)y + B(t)v + s(t), \qquad y(0) = y_0 \end{aligned}$$

is minimized. Here Q, R, S, r, z, A, B, and s are functions of appropriate dimensionality defined on the interval $[0,T]$. We show how the algorithms of §§2 and 3 can be adapted to this case.

A recent paper by Chang, Chang, and Luh [3] proposes a parallel algorithm for

(1)–(2) which has a similar flavor to the methods discussed here. They use an approach that is similar to multiple shooting for two-point boundary value problems, defining the state variables for certain equally spaced values of the index i as unknowns in a reduced optimization problem. The Hessian of this problem can be formed in parallel, and Newton steps are calculated by using cyclic reduction. The main differences from our algorithms are that, in [3], the problem is partitioned at the level of the nonlinear problem, and the algorithm consists of two distinct levels (whereas our methods can be implemented in up to $\lceil \log_2 N \rceil$ levels). We believe that exploiting parallelism at the level of the linear algebra allows more flexibility to enhance the algorithm at the nonlinear level (for example, by modifying it to handle constraints, to ensure global convergence, or to use approximate Hessians), without causing complications.

2. Dynamic programming. It is known that linear systems with dimension M and bandwidth K can be solved by using Gaussian elimination in $O(MK^2)$ operations. For the system (10)–(11), this translates to $O(N(m^3+n^3))$ operations. The dynamic programming approaches described below have just such a complexity bound; in fact, they can be thought of as specialized block elimination algorithms for this system.

One algorithm, described by Polak in [10], proceeds by first eliminating the v_i using equation (6). Substitution in (7)–(9) yields the system

$$\hat{Q}_i y_i + \hat{A}_i^T q_{i+1} - q_i + \hat{t}_i = 0, \qquad i = 1, \cdots, N, \tag{15}$$

$$-y_{i+1} + \hat{A}_i y_i + \hat{J}_i q_{i+1} + \hat{s}_i = 0, \qquad i = 1, \cdots, N, \tag{16}$$

where

$$\begin{aligned} \hat{A}_i &= A_i - B_i(S_i)^{-1}R_i^T, \\ \hat{J}_i &= -B_i(S_i)^{-1}B_i^T, \\ \hat{Q}_i &= Q_i - R_i(S_i)^{-1}R_i^T, \\ \hat{t}_i &= z_i - R_i(S_i)^{-1}r_i, \\ \hat{s}_i &= s_i - B_i(S_i)^{-1}r_i. \end{aligned}$$

Then, a Riccati substitution is made for q_i; that is, we seek matrices K_i and vector b_i such that

$$q_i = K_i y_i + b_i. \tag{17}$$

Clearly, from (8), $K_{N+1} = Q_{N+1}$ and $b_{N+1} = z_{N+1}$. By combining (15), (16), and (17), we can obtain expressions of the form

$$W_i y_i + w_i = 0, \qquad i = 1, \cdots, N, \tag{18}$$

where W_i depends on K_i and K_{i+1}, and w_i depends on b_i and b_{i+1}. Since (18) must be satisfied for all values of y_i, we deduce that $W_i = 0$ and $w_i = 0$. This yields recurrence relations for the K_i and b_i, which give rise to the following algorithm.

Algorithm RI

$K_{N+1} \leftarrow Q_{N+1}$, $b_{N+1} \leftarrow z_{N+1}$;

for $i = N, \cdots, 2$

$$\begin{aligned} K_i &\leftarrow \hat{A}_i^T[I - K_{i+1}\hat{J}_i]^{-1}K_{i+1}\hat{A}_i + \hat{Q}_i, \\ b_i &\leftarrow \hat{A}_i^T[I - K_{i+1}\hat{J}_i]^{-1}(K_{i+1}\hat{s}_i + b_{i+1}) + \hat{t}_i; \end{aligned}$$

$y_1 \leftarrow 0$;
for $i = 1, \cdots, N$

$$y_{i+1} \leftarrow (I - K_{i+1}\hat{J}_i)^{-T}[\hat{A}_i y_i + \hat{s}_i + \hat{J}_i b_{i+1}];$$

recover q_{i+1} from (17); recover v_i from (6).

Note that the factorizations of the matrices $(I - K_{i+1}\hat{J}_i)$ (needed in the first loop) can be re-used as factors of $(I - K_{i+1}\hat{J}_i)^T$ in the second loop.

In counting the higher-order terms in the operation count for this algorithm, we assume that advantage is taken of symmetry, where appropriate (K_i, $\hat{J}_i$, and $\hat{Q}_i$ are all symmetric), and that additions, multiplications, and divisions each count as one operation. We find that approximately

$$N(7n^3 + 4n^2 m + 4nm^2 + \tfrac{1}{3}m^3) \tag{19}$$

operations are needed for Algorithm RI.

We note for later reference that this algorithm can be trivially modified if a term involving q_{i+1} is introduced into equation (9). If (9) is replaced by

$$s_i - y_{i+1} + A_i y_i + B_i v_i + J_i q_{i+1} = 0, \tag{20}$$

we need only modify the definition of $\hat{J}_i$ to $J_i - B_i(S_i)^{-1}B_i^T$. The remainder of the algorithm is unchanged.

From the outset, RI depends on the nonsingularity of the matrices S_i. Although an analogous property is usually assumed to hold in the *continuous-time* problem (as we note in §7), Dunn and Bertsekas have noted that in the *discrete-time* case, this property may not hold even when (3) has a unique, finite solution. The algorithm described in [5] will, on the other hand, produce a solution for (3) whenever one exists. It proceeds by finding matrices θ_i and Γ_i, and vectors β_i and γ_i, such that

$$\begin{aligned} q_i &= \theta_i y_i + \beta_i, \qquad i = 1, \cdots, N+1 \\ v_i &= \Gamma_i y_i + \gamma_i, \qquad i = 1, \cdots, N. \end{aligned}$$

Substitution into the equations (6)–(9), and some manipulation, gives the following algorithm.

Algorithm DP

$\theta_{N+1} \leftarrow Q_{N+1}$; $\beta_{N+1} \leftarrow z_{N+1}$;

for $i = N, \cdots, 1$

$$\begin{aligned} \Gamma_i &\leftarrow -[S_i + B_i^T\theta_{i+1}B_i]^{-1}(R_i^T + B_i^T\theta_{i+1}A_i), \\ \gamma_i &\leftarrow -[S_i + B_i^T\theta_{i+1}B_i]^{-1}(r_i + B_i^T(\theta_{i+1}s_i + \beta_{i+1})), \\ \theta_i &\leftarrow (Q_i + A_i^T\theta_{i+1}A_i) + (R_i + A_i^T\theta_{i+1}B_i)\Gamma_i \qquad (i \neq 1), \\ \beta_i &\leftarrow A_i^T\theta_{i+1}(B_i\gamma_i + s_i) + A_i^T\beta_{i+1} + z_i + R_i\gamma_i \qquad (i \neq 1). \end{aligned}$$

The steps v_i and y_i can then be recovered in the following loop:

$y_1 = 0$;

for $i = 1, \cdots, N$
$v_i \leftarrow \Gamma_i y_i + \gamma_i$
$y_{i+1} \leftarrow A_i y_i + B_i v_i + s_i$.

The following operation count is obtained:

$$N(3n^3 + 5n^2 m + 3nm^2 + \tfrac{1}{3}m^3) + O(N(m^2 + n^2)). \tag{21}$$

As they stand, these algorithms can only exploit parallelism and vectorization *within* each iteration (i.e., in the matrix multiplications and factorizations), and hence reasonable efficiency can be expected only when m and n are fairly large. When the matrix (10) is sparse within its band, the situation is complicated further by a need for parallel sparse linear algebra algorithms. Moreover, the number of stages N is typically quite large. This is the motivation for considering parallelism *across the loop*, which we proceed to do in the next section.

3. Partitioned dynamic programming. Here we describe variants of the algorithms above which are more arithmetically expensive, but more amenable to parallel implementation. The problem is broken up into P partitions, where each adjacent pair of partitions is separated by a single stage. Within each partition, a variant of either DP or RI is used to express the "internal" variables for each partition in terms of the variables in the adjoining separator stages. A reduced problem consisting of P stages is then formed, and the process is repeated, possibly recursively. A detailed analysis of the possibilities appears in §5.

3.1. Partitioned version of DP. The initial partitioning is done by choosing "separator indices" $I_1, I_2, \cdots, I_P, I_{P+1}$ that lie in the range $1, 2, \cdots, N+1$ and satisfy the relationships

$$I_1 = 1; \qquad I_{j+1} \geq I_j + 2, \qquad (j = 1, \cdots, P); \qquad I_{P+1} = N + 1.$$

It follows from these requirements that P cannot exceed $(N+1)/2$. Usually, the separator indices are chosen to be spaced approximately equally, in which case they are approximately $(N+1)/P$ stages apart. The variables q_{I_j}, y_{I_j} and v_{I_j}, $j = 1, \cdots, P$ will be referred to as "separator variables." For these, it is convenient to use the notation

$$\tilde{q}_j \stackrel{\text{def}}{=} q_{I_j}; \qquad \tilde{y}_j \stackrel{\text{def}}{=} y_{I_j}; \qquad \tilde{v}_j \stackrel{\text{def}}{=} v_{I_j}. \tag{22}$$

These variables are the unknowns in the reduced system. Each partition consists of the indices strictly between two separators; partition j will consist of stages $I_j + 1$ to $I_{j+1} - 1$ inclusive.

Within each partition, we start by seeking matrices θ_i, Γ_i, D_i, and F_i and vectors β_i and γ_i such that

$$q_i = \theta_i y_i + D_i q_{I_{j+1}} + \beta_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1, \tag{23}$$

$$v_i = \Gamma_i y_i + F_i q_{I_{j+1}} + \gamma_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1. \tag{24}$$

Proceeding as in DP, using the equations (6)–(9) and (23)–(24), we can compute these matrices and vectors as follows.

Algorithm PDP

$$\theta_{I_{j+1}} \leftarrow 0; \qquad \beta_{I_{j+1}} \leftarrow 0; \qquad D_{I_{j+1}} \leftarrow I \tag{25}$$

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j + 1$

$$\Gamma_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1}(R_i^T + B_i^T \theta_{i+1} A_i), \tag{26}$$

$$\gamma_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1}(r_i + B_i^T(\theta_{i+1} s_i + \beta_{i+1})), \tag{27}$$

$$F_i \leftarrow -[S_i + B_i^T \theta_{i+1} B_i]^{-1} B_i^T D_{i+1}, \tag{28}$$

$$D_i \leftarrow (A_i + B_i \Gamma_i)^T D_{i+1}, \tag{29}$$

$$\theta_i \leftarrow (Q_i + A_i^T \theta_{i+1} A_i) + (R_i + A_i^T \theta_{i+1} B_i)\Gamma_i, \tag{30}$$

$$\beta_i \leftarrow A_i^T \theta_{i+1}(B_i \gamma_i + s_i) + A_i^T \beta_{i+1} + z_i + R_i \gamma_i. \tag{31}$$

We now aim to construct a reduced system in which only the separator variables are unknown. One equation can be deduced from (9) with $i = I_{j+1} - 1$, namely,

$$s_{I_{j+1}-1} + A_{I_{j+1}-1} y_{I_{j+1}-1} + B_{I_{j+1}-1} v_{I_{j+1}-1} = \tilde{y}_{j+1}.$$

The variables $v_{I_{j+1}-1}$ and $y_{I_{j+1}-1}$ can be eliminated by substituting from (24) with $i = I_{j+1} - 1$, and then (9) with $i = I_{j+1} - 2$. This substitution yields an equation in which the unknowns on the left-hand side are $y_{I_{j+1}-2}$ and $v_{I_{j+1}-2}$. This process is repeated for $i = I_{j+1} - 2$ down to $i = I_j$, at which point $\tilde{y}_j$ and $\tilde{v}_j$ appear. The resulting equation is

$$\tilde{A}_j \tilde{y}_j + \tilde{B}_j \tilde{v}_j + \tilde{J}_j \tilde{q}_{j+1} + \tilde{s}_j = \tilde{y}_{j+1}, \tag{32}$$

where

$$\begin{aligned}
\tilde{A}_j &= D_{I_j+1}^T A_{I_j}, \\
\tilde{B}_j &= D_{I_j+1}^T B_{I_j}, \\
\tilde{J}_j &= \sum_{i=I_j+1}^{I_{j+1}-1} D_{i+1}^T B_i F_i, \\
\tilde{s}_j &= \sum_{i=I_j+1}^{I_{j+1}-1} D_{i+1}^T (B_i \gamma_i + s_i) + D_{I_j+1}^T s_{I_j}.
\end{aligned}$$

A second equation is derived by setting $i = I_j$ in (7), $i = I_j + 1$ in (23), and $i = I_j$ in (9):

$$\tilde{t}_j + \tilde{R}_j \tilde{v}_j + \tilde{Q}_j \tilde{y}_j - \tilde{q}_j + \tilde{A}_j^T \tilde{q}_{j+1} = 0, \tag{33}$$

where

$$\begin{aligned}
\tilde{t}_j &= z_{I_j} + A_{I_j}^T(\theta_{I_j+1} s_{I_j} + \beta_{I_j+1}), \\
\tilde{R}_j &= R_{I_j} + A_{I_j}^T \theta_{I_j+1} B_{I_j}, \\
\tilde{Q}_j &= Q_{I_j} + A_{I_j}^T \theta_{I_j+1} A_{I_j}.
\end{aligned}$$

A third equation is derived by setting $i = I_j$ in (6), $i = I_j + 1$ in (23), and $i = I_j$ in (9):

$$\tilde{r}_j + \tilde{R}_j^T \tilde{y}_j + \tilde{S}_j \tilde{v}_j + \tilde{B}_j^T \tilde{q}_{j+1} = 0, \tag{34}$$

where

$$\tilde{r}_j = r_{I_j} + B_{I_j}^T(\theta_{I_j+1}s_{I_j} + \beta_{I_j+1}), \qquad \tilde{S}_j = S_{I_j} + B_{I_j}^T\theta_{I_j+1}B_{I_j},$$

and $\tilde{R}_j$ and $\tilde{B}_j$ are as defined above.

Two additional equations, for the first and last partitions, complete the system. From the initial condition, we have

$$\tilde{y}_1 = s_0, \tag{35}$$

while from (8), using $I_{P+1} = N + 1$, we have

$$\tilde{q}_{P+1} = Q_{N+1}\tilde{y}_{P+1} + z_{N+1}. \tag{36}$$

If the separator variables are ordered as

$$(\tilde{q}_1, \tilde{y}_1, \tilde{v}_1, \tilde{q}_2, \tilde{y}_2, \tilde{v}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}),$$

and the right-hand sides are ordered as

$$(-s_0, -\tilde{t}_1, -\tilde{r}_1, -\tilde{s}_1, \cdots, -\tilde{t}_P, -\tilde{r}_P, -\tilde{s}_P, -z_{N+1}),$$

the coefficient matrix for (32)–(34), (35), and (36) has the following form:

$$\begin{bmatrix}
0 & -I & & & & & & & & & \\
-I & \tilde{Q}_1 & \tilde{R}_1 & \tilde{A}_1^T & & & & & & & \\
 & \tilde{R}_1^T & \tilde{S}_1 & \tilde{B}_1^T & & & & & & & \\
 & \tilde{A}_1 & \tilde{B}_1 & \tilde{J}_1 & -I & & & & & & \\
 & & & -I & \tilde{Q}_2 & \tilde{R}_2 & \tilde{A}_2^T & & & & \\
 & & & & \tilde{R}_2^T & \tilde{S}_2 & \tilde{B}_2^T & \ddots & & & \\
 & & & & & \ddots & \ddots & \ddots & & & \\
 & & & & & & & & \tilde{R}_P^T & \tilde{S}_P & \tilde{B}_P^T & \\
 & & & & & & & & \tilde{A}_P & \tilde{B}_P & \tilde{J}_P & -I \\
 & & & & & & & & & & -I & Q_{N+1}
\end{bmatrix}. \tag{37}$$

In comparison with DP, the main additional expenses in this partitioned algorithm involve calculation of the matrices F_i, D_i, $\tilde{J}_j$, $\tilde{A}_j$, $\tilde{B}_j$, $\tilde{Q}_j$, and $\tilde{R}_j$. Savings can be made in the first iteration of each loop (25)–(31) by taking note of the initial values of θ_{I_j} and D_{I_j}. Again, if lower-order terms are ignored, the operation count is approximately

$$(N - P)[6n^3 + 9n^2m + 5nm^2 + \tfrac{1}{3}m^3]. \tag{38}$$

All of the above constitutes phase I of the algorithm. The remaining phases consist of solving the reduced system (phase II) and recovering the internal states y_i from (9) and the internal controls and Lagrange multipliers v_i and q_i from (23) and (24) (phase III). The total operation count for phase III is $O(N(m^2 + n^2))$, so its cost is dominated by that of phase I unless m and n are very small. We do not take account of phase III in the timing analyses below. Phase II takes $O(P(m^3 + n^3))$ and, therefore, may be significant; this situation will be discussed further.

Unfortunately, in doing the partitioning of the dynamic programming algorithm, we lose the property that the inverses of the operators $[S_i + B_i^T \theta_{i+1} B_i]$ exist for all i. This phenomenon is similar to rank deficiency in the submatrices obtained by partitioning nonsingular banded linear systems (see, for example, [13]). The robustness of a parallel code could therefore be improved by including the algorithm of Wright [14] as a backup, in case PDP fails.

3.2. Partitioned version of RI. We now specify a partitioned version of RI applied to the equations (6), (7), (8), and (20). The controls can be eliminated as before, and the system (15)–(16) can be obtained, with the appropriate modification to the definition of $\hat{J}_i$. For i in the range I_j to $I_{j+1} - 1$, we make the following "Riccati" substitution for q_i:

$$q_i = K_i y_i + L_i^T \tilde{q}_{j+1} + b_i. \tag{39}$$

Manipulating the equations in the usual way, we obtain the following scheme for calculating K_i, L_i, and b_i.

Algorithm PRI (part 1)

$K_{I_{j+1}} \leftarrow 0$, $L_{I_{j+1}} \leftarrow I$, $b_{I_{j+1}} \leftarrow 0$;

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j$

$$\begin{aligned} K_i &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} K_{i+1} \hat{A}_i + \hat{Q}_i, \\ L_i^T &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} L_{i+1}^T, \\ b_i &\leftarrow \hat{A}_i^T [I - K_{i+1}\hat{J}_i]^{-1} (K_{i+1}\hat{s}_i + b_{i+1}) + \hat{t}_i. \end{aligned}$$

Again, we need to define a reduced system. One equation in this system can be found by setting $i = I_j$ in the formula (39), giving

$$\tilde{q}_j = K_{I_j} \tilde{y}_j + L_{I_j}^T \tilde{q}_{j+1} + b_{I_j}. \tag{40}$$

The second equation is derived by seeking matrices H_i and M_i, and vectors h_i, such that for $i = I_j, \cdots, I_{j+1} - 1$,

$$H_i y_i + M_i \tilde{q}_{j+1} - \tilde{y}_{j+1} + h_i = 0.$$

This can be achieved by the following loop.

Algorithm PRI (part 2)

$M_{I_{j+1}} \leftarrow 0$, $H_{I_{j+1}} \leftarrow I$, $h_{I_{j+1}} \leftarrow 0$.

for $i = I_{j+1} - 1, I_{j+1} - 2, \cdots, I_j$

$$\begin{aligned} H_i &\leftarrow H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} \hat{A}_i, \\ M_i &\leftarrow M_{i+1} + H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} \hat{J}_i L_{i+1}, \\ h_i &\leftarrow h_{i+1} + H_{i+1}(I - \hat{J}_i K_{i+1})^{-1} (\hat{J}_i b_{i+1} + \hat{s}_i). \end{aligned}$$

It is easy to see that $H_i = L_i$ for all i. Of course, parts 1 and 2 can be combined in the same loop. Setting $i = I_j$, we obtain the second equation of the reduced system:

$$L_{I_j} \tilde{y}_j + M_{I_j} \tilde{q}_{j+1} - \tilde{y}_{j+1} + h_{I_j} = 0. \tag{41}$$

It is important in this context to distinguish between the $K_{I_{j+1}}$ used as an initial value for the partition-j calculations (that is, $K_{I_{j+1}} = 0$), and the $K_{I_{j+1}}$ obtained as the end product of the partition-$(j+1)$ calculations. In constructing the reduced system, we assume the latter. Below, the notation $\tilde{K}_j$ is used for K_{I_j}, and so on, and we again use the additional equations (35)–(36) for the first and last partitions.

Ordering unknowns as

$$(\tilde{q}_1, \tilde{y}_1, \tilde{q}_2, \tilde{y}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}),$$

and the right-hand side as

$$(-s_0, -\tilde{b}_1, -\tilde{h}_1, -\tilde{b}_2, \cdots, -\tilde{b}_P, -\tilde{h}_P, -z_{N+1}),$$

we get as the coefficient matrix for the reduced system

$$\begin{bmatrix} 0 & -I & & & & & \\ -I & \tilde{K}_1 & \tilde{L}_1^T & & & & \\ & \tilde{L}_1 & \tilde{M}_1 & -I & & & \\ & & \ddots & \ddots & & & \\ & & & & \tilde{L}_P & \tilde{M}_P & -I \\ & & & & & -I & Q_{N+1} \end{bmatrix}. \tag{42}$$

In counting operations for the algorithm so far, we note that S_i^{-1}, $\hat{J}_i$, $\hat{A}_i$, and $\hat{Q}_i$ must be calculated for all N stages. Savings can be made by noting that the first iteration of each of the loops above (that is, calculation of $K_{I_{j+1}-1}$, $L_{I_{j+1}-1}$, $M_{I_{j+1}-1}$) is virtually free; hence we multiply the cost for each iteration of these loops only by $(N-P)$. The higher-order terms sum to

$$N[15n^3 + 3n^2m + 5nm^2 + \tfrac{1}{3}m^3] - P[15n^3]. \tag{43}$$

3.3. Recursive implementation. The two partitioned algorithms above can be used in conjunction with the serial algorithms of §2 to devise recursive methods for solving (3). We note the following two facts:

- both the serial and partitioned dynamic programming algorithms have lower operation counts than the corresponding Riccati-based algorithms, but
- Algorithms RI and PRI can be used when (9) has been replaced by (20).

If we apply PDP to solve the original system (as we should, because of the first fact), we obtain a reduced system which contains equations of the form (20) instead of (9). Hence, because of the second fact, the reduced system, and any other reduced systems which arise at subsequent levels of recursion, can be solved only by using RI or PRI. In general, the optimal strategy for solving (3) consists of an application of PDP, followed by repeated applications of PRI (on fewer and fewer processors), followed by an application of RI on a single processor. Recall that we have the constraint that the size of the reduced system must decrease by a factor of at least 2 at each level of recursion; hence the theoretical limit on the number of levels is $\lceil \log_2 N \rceil$. In §5, we discuss "optimal" strategies for multiprocessor implementation.

The variant of the method in which the maximal number of levels is used is, effectively, a specialized form of cyclic reduction. It may be worth considering implementation of this variant on vector architectures, particularly when n and m are very small, so that factorizations involving individual blocks cannot be vectorized efficiently. We do not pursue this issue here, but refer the reader to Wright [15], in which cyclic-reduction–like algorithms for linear systems arising from general two-point boundary value problems are implemented and discussed.

4. Constrained problems. Usually, additional constraints are applied to the controls and/or states in the problem (1)–(2). The partitioned methods of the previous section can be extended in a straightforward way to the control-constrained case; this is the topic we discuss in this section.

Suppose that in (1)–(2) we have the additional constraints

$$g_i(u_i) \leq 0, \qquad i = 1, \cdots, N. \tag{44}$$

Often these constraints are simple bounds, or Cartesian products of simple geometric shapes such as spheres or cones (see Gawande and Dunn [6]). Algorithms such as two-metric gradient projection, or sequential quadratic programming with an active set strategy, may then be applied to solve (1), (2), and (44). It is not our aim to discuss the properties of these methods here, but rather to focus on the main computational tasks and their implementation. Both methods give rise to subproblems of the form (3), with the additional constraints

$$G_i v_i + g_i = 0, \tag{45}$$

where the vector g_i in (45) may contain only a subset of the components of $g_i(u_i)$ from (44). The terms in the objective function of (3) must be defined in terms of the modified Lagrangian

$$\mathcal{L}(u, x, p) = \ell_{N+1}(x_{N+1}) + \sum_{i=1}^{N} \ell_i(x_i, u_i) - \sum_{i=1}^{N} p_{i+1}^T (x_{i+1} - f_i(x_i, u_i)) + \sum_{i=1}^{N} \mu_i^T g_i(u_i).$$

First-order necessary conditions for (3), (45) yield the equations (7), (8), (9), (45), while (6) is replaced by

$$r_i + R_i^T y_i + S_i v_i + B_i^T q_{i+1} + G_i^T \mu_i = 0, \qquad i = 1, \cdots, N. \tag{46}$$

Standard null-space techniques can now be used to eliminate the μ_i and obtain a system of the form (6)–(9). Assuming without loss of generality that the G_i have full row rank, we can define orthogonal matrices U_i and upper triangular matrices T_i such that

$$U_i G_i^T = \begin{bmatrix} T_i \\ 0 \end{bmatrix}.$$

Partitioning U_i in the obvious way as

$$U_i = \begin{bmatrix} U_{i1} \\ U_{i2} \end{bmatrix},$$

and writing

$$\bar{v}_{i1} = U_{i1} v_i, \qquad \bar{v}_{i2} = U_{i2} v_i,$$

we obtain by substitution in (7), (9), (20), (45), and (46) the system

$$\bar{r}_{i2} + \bar{R}_{i2}^T y_i + \bar{S}_{i22} \bar{v}_{i2} + \bar{B}_{i2}^T q_{i+1} = 0, \qquad i = 1, \cdots, N, \tag{47}$$

$$\bar{t}_i + \bar{R}_{i2} \bar{v}_{i2} + Q_i y_i - q_i + A_i^T q_{i+1} = 0, \qquad i = 1, \cdots, N, \tag{48}$$

$$z_{N+1} + Q_{N+1} y_{N+1} - q_{N+1} = 0, \tag{49}$$

$$\bar{s}_i - y_{i+1} + A_i y_i + \bar{B}_{i2} \bar{v}_{i2} = 0, \qquad i = 1, \cdots, N. \tag{50}$$

Here $\bar{R}_{i2}$, $\bar{B}_{i2}$, $\bar{S}_{i22}$ are parts of the transformed matrices

$$\bar{R}_i = U_i R_i = \begin{bmatrix} \bar{R}_{i1} \\ \bar{R}_{i2} \end{bmatrix}, \quad \bar{B}_i = U_i B_i = \begin{bmatrix} \bar{B}_{i1} \\ \bar{B}_{i2} \end{bmatrix}, \quad \bar{S}_i = U_i S_i U_i^T = \begin{bmatrix} \bar{S}_{i11} & \bar{S}_{i12} \\ \bar{S}_{i21} & \bar{S}_{i22} \end{bmatrix},$$

while

$$\begin{aligned} \bar{r}_{i2} &= U_{i2} r_i + \bar{S}_{i21} \bar{v}_{i1}, \\ \bar{t}_i &= z_i + \bar{R}_{i1} \bar{v}_{i1}, \\ \bar{s}_i &= s_i + \bar{B}_{i1} \bar{v}_{i1}. \end{aligned}$$

The vectors $\bar{v}_{i1}$ and μ_i can be found from

$$\begin{aligned} \bar{v}_{i1} &= -T_i^{-T} g_i, \\ \mu_i &= -T_i^{-1} U_{i1} [r_i + R_i y_i + S_i \bar{v}_i + B_i^T q_{i+1}]. \end{aligned}$$

Clearly, the serial and parallel methods discussed in §§2 and 3 can be applied to solve (47)–(50), and this is typically the most expensive part of each major iteration. However, gradient projection algorithms for (1)–(2) perform two other significant calculations at each iteration. These are solution of the adjoint equation

$$\begin{aligned} p_{N+1} &= \nabla \ell_{N+1}(x_{N+1}), \\ p_i &= \frac{\partial f_i}{\partial x_i}^T p_{i+1} + \frac{\partial \ell_i}{\partial x_i}, \qquad i = N, \cdots, 1 \end{aligned} \tag{51}$$

(which is needed as part of the calculation of $\nabla F(u)$), and the evaluation of the nonlinear recurrence (2) for a given set of controls u_i. The technique for parallelizing (51) is essentially the same as that used in §3. The stages are partitioned into P groups, and within partition j, matrices E_i and vectors e_i are sought such that

$$p_i = E_i p_{I_{j+1}} + e_i, \qquad i = I_{j+1} - 1, \cdots, I_j + 1.$$

Recurrence relations for E_i and e_i can be derived.

When the state equations f_i are all linear, an identical technique can be used to speed up evaluation of (2). Here, in partition j, we seek H_i and h_i such that

$$x_i = H_i x_{I_j} + h_i, \qquad i = I_j + 1, \cdots, I_{j+1} - 1. \tag{52}$$

When the f_i are nonlinear, it is impossible to derive such linear relationships between the x_i which are the basis of the speedup techniques described in this paper. It is possible, in principle, to develop nonlinear analogs of (52), but implementation of these would require nonnumerical computing techniques which are outside the scope of this paper. One way around this bottleneck could be to use a two-level algorithm. At each outer iteration, the "IQP variant" of sequential quadratic programming is applied to (1), (2), and (44) to obtain a subproblem (3), with the additional linear inequality constraints

$$G_i v_i + g_i \leq 0, \qquad i = 1, \cdots, N.$$

This linearized problem (with linearized state equations) can then be solved by using two-metric gradient projection.

For problems with additional constraints on the states, quite different techniques from those above are required. This requirement is discussed in Psiaki and Park [11], who propose a recursive method involving variable elimination and pairwise combination of stages. We note that their method can be extended to allow merging of any number of successive stages at each level of recursion (not just 2), but refer the reader to [11] for details.

5. Optimal multiprocessor implementation. The operation count expressions derived in earlier sections can be used to give some insight into optimizing the number of levels of recursion, and optimizing the allocation of processors within each level for the algorithm of §3.3. The results of this section apply to the unconstrained problem, and to the cost of solving the reduced system (47)–(50) for the constrained problem. We start by introducing new notation for the operation counts of §§2 and 3. Since it is usually true that $m \leq n$, we write

$$m = \alpha n,$$

where $\alpha \in (0, 1]$. Then, scaling in each place by n^3, (19), (21), (38), and (43) can be replaced by

$$\begin{aligned} \text{Algorithm RI:} &\quad NA_\alpha, \\ \text{Algorithm DP:} &\quad NB_\alpha, \\ \text{Algorithm PRI:} &\quad NC_\alpha - PD_\alpha, \\ \text{Algorithm PDP:} &\quad (N-P)E_\alpha, \end{aligned}$$

where

$$\begin{aligned} A_\alpha &= 7 + 4\alpha + 4\alpha^2 + \tfrac{1}{3}\alpha^3, \\ B_\alpha &= 3 + 5\alpha + 3\alpha^2 + \tfrac{1}{3}\alpha^3, \\ C_\alpha &= 15 + 3\alpha + 5\alpha^2 + \tfrac{1}{3}\alpha^3, \\ D_\alpha &= 15, \\ E_\alpha &= 6 + 9\alpha + 5\alpha^2 + \tfrac{1}{3}\alpha^3. \end{aligned}$$

In the remainder of this section, quantities denoted by τ can be converted to absolute runtimes by multiplying by $n^3\delta$, where δ is the time required for one floating-point operation. For simplicity, however, we refer to the τ themselves as "runtimes."

5.1. Shared-memory machines. We start by discussing implementation on shared-memory multiprocessors. This is the simplest case, as we do not need to take account of interprocessor communication costs, so the operation counts above will give a reasonable indication of the elapsed time needed to solve a problem (3). However, when n is small, we also need to take into account the lower-order terms that were discarded earlier. When the total amount of data in the problem ($O(Nn^2)$) is large, the presence of hierarchical memory may also affect the times. This effect should at least be consistent as N increases, since in all algorithms, the data is processed in two or three sequential sweeps. Other costs which are ignored are costs of array indexing and manipulation of data structures, particularly the manipulation necessary to build up the reduced systems. The ratio of these costs to the arithmetic costs varies like $1/n$, so they may be significant when n is small.

On a shared-memory machine, the number of available processors P will typically be substantially smaller than the number of stages N. It seems reasonable, therefore, to consider a σ-level algorithm, in which PDP is first performed on all P processors, and then the reduced systems are solved by using PRI on fewer and fewer processors, until finally at the lowest level, RI is performed on a single processor. In general, at level k, a problem with P_k stages is solved on P_{k-1} processors ($P_\sigma = N$, $P_{\sigma-1} = P$, $P_0 = 1$). The runtime is proportional to

$$\tau_\sigma = \frac{N-P}{P} E_\alpha + \sum_{i=1}^{\sigma-2} \frac{P_{\sigma-i} C_\alpha - P_{\sigma-1-i} D_\alpha}{P_{\sigma-1-i}} + P_1 A_\alpha. \tag{53}$$

TABLE 1
Theoretical optimal processor allocations and scaled runtimes, where $P = 16$, $\alpha = .5$ *and* $N \geq 32$.

σ	P_1	P_2	P_3	P_4	P_5	$\tau_{\sigma,\text{opt}} - \tau_{DP}$
2	16.0					161
3	5.3	16.0				91.9
4	3.7	7.7	16.0			81.1
5	3.1	5.3	9.2	16.0		78.4
6	2.8	4.3	6.6	10.3	16.0	78.1

Minimizing τ_σ with respect to $P_{\sigma-2}, \cdots, P_1$, we find that the optimal values for the P_k are

$$P_{k,\text{opt}} = \left(\frac{C_\alpha}{A_\alpha}\right)^{(\sigma-1-k)/(\sigma-1)} P^{k/(\sigma-1)}, \qquad k = 1, \cdots, \sigma - 2, \tag{54}$$

giving an optimal runtime of

$$\tau_{\sigma,\text{opt}} = \left(\frac{N}{P} - 1\right) E_\alpha + (\sigma - 1) C_\alpha \left(\frac{A_\alpha P}{C_\alpha}\right)^{1/(\sigma-1)} - (\sigma - 2) D_\alpha. \tag{55}$$

To illustrate this analysis, we insert some typical values of the parameters. Setting $\alpha = .5$ and $P = 16$, and assuming $N \geq 32$, values of $P_{k,\text{opt}}$ and $\tau_{\sigma,\text{opt}}$ are given in Table 1. We actually tabulate $\tau_{\sigma,\text{opt}} - \tau_{DP}$, where $\tau_{DP} = ((N/P) - 1)E_\alpha$, (that is, we exclude that part of the runtime expression which is independent of σ). The theoretical minimizing σ is 5.62, for which an adjusted runtime of 78.0 is obtained. Because we have the constraints that σ and the P_k are integers, that $P_k \geq 2P_{k-1}$, and that, preferably, P_k/P_{k-1} are integers, we use Table 1 as a guide and look for a nearby feasible schedule. The schedule $\sigma = 4$, $P_1 = 4$, $P_2 = 8$, $P_3 = 16$ produces a near-optimal adjusted runtime of 81.3. The five-stage schedule $P_1 = 2$, $P_2 = 4$, $P_3 = 8$, $P_4 = 16$ gives a time of 81.8. (We call this a "cyclic reduction" schedule, since after the initial execution of PDP on 16 processors, the size of the reduced system is halved at each level, as occurs in cyclic reduction applied to block-tridiagonal systems. See, for example, Golub and Van Loan [7].)

Finally we note that for systems with small numbers of processors, the scheduling above is in the nature of fine-tuning, since most of the work takes place at the top level in PDP. As mentioned, we have discounted the time required for this phase from the calculations in Table 1. In this example, $((N/P) - 1)E_\alpha$ would be 177 when $N = 256$ and 1498 when $N = 2048$.

5.2. Message-passing machines. We turn now to implementation on message-passing architectures. Here, we need to take into account the communication overhead needed in moving from one level of the algorithm to the next, and the availability of more processors than are typically found on shared-memory machines. Fortunately, the communication pattern is quite regular.

We assume that, at the top level of the algorithm, each of the P processors contains the data needed to execute its share of PDP. Processor j needs to have A_k, B_k, Q_k, R_k, S_k, s_k, z_k, and r_k for $k = I_j, \cdots, I_{j+1} - 1$. After doing its computation, this processor contributes the blocks $\tilde{A}_j$, $\tilde{B}_j$, $\tilde{J}_j$, $\tilde{Q}_j$, $\tilde{R}_j$, $\tilde{S}_j$, $\tilde{s}_j$, $\tilde{t}_j$, and $\tilde{r}_j$ to the reduced system. If processor j is not among the $P_{\sigma-2}$ processors which will be used

at the next level, it needs to send these blocks to another processor. The total length of the message will be $n^3 F_\alpha$, where $F_\alpha = (3 + 2\alpha + \alpha^2)/n + (2 + \alpha)/n^2$ words.

During the final phase, in which values of y_k, v_k, and q_k are being recovered, communication takes place in the reverse direction: processor j needs to know the values of $\tilde{q}_{j+1}$ and $\tilde{y}_j$. These may need to be sent from one of the $P_{\sigma-2}$ processors which were used at the next lower level. A total of $(P_{\sigma-1} - P_{\sigma-2})$ messages of length $n^3 G_\alpha$ words is involved, where $G_\alpha = 2/n^2$.

In general, communication between level $i + 1$ and level i requires a total of $P_i - P_{i-1}$ messages of length $n^3 F_\alpha$ to be sent (concurrently) during the first phase, and $P_i - P_{i-1}$ messages of length $n^3 G_\alpha$ to be sent (concurrently) during the second phase.

We assume that the time required to send a message of length M eight-byte words can be approximated by the formula

$$\gamma + \beta M, \tag{56}$$

where M is the number of words. This timing model has often been used for transfer of data between adjacent (i.e., directly connected) processors. However, newer hypercubes, such as the Intel iPSC/2 and the Ncube, use a routing mechanism which makes (56) a reasonable timing model for transfer between *any* two processors.

By modifying (53), we model the total runtime for a distributed-memory machine as

$$\tau_\sigma = \left(\frac{N}{P_{\sigma-1}} - 1\right) E_\alpha + \sum_{i=1}^{\sigma-2} \frac{P_{\sigma-i} C_\alpha}{P_{\sigma-1-i}} + P_1 A_\alpha - (\sigma - 2) D_\alpha + (\sigma - 1) H_\alpha, \tag{57}$$

where

$$H_\alpha = 2\frac{\gamma}{n^3 \delta} + (F_\alpha + G_\alpha)\frac{\beta}{\delta}.$$

Note that the new term is independent of $P_1, \cdots P_{\sigma-1}$ and so, for fixed σ, the optimal processor allocation will be the same as in the shared-memory case (54). Again we focus on the case in which the number of available processors P is less than $N/2$. For the optimal processor schedule (54), the runtime will be

$$\tau_{\sigma,\text{opt}} = \left(\frac{N}{P} - 1\right) E_\alpha + (\sigma - 1) C_\alpha \left(\frac{A_\alpha P}{C_\alpha}\right)^{1/(\sigma-1)} - (\sigma - 2) D_\alpha + (\sigma - 1) H_\alpha.$$

Again, we use some reasonable parameter values to obtain a feeling for performance. The experiments of Dunigan [4, Tables 3 and 6] show that, for double-precision arithmetic, approximate values of γ, β, and δ for the Ncube are

$$\gamma = 384\mu s, \qquad \beta = 20.8\mu s, \qquad \delta = 7.8\mu s.$$

We give predicted results for this machine, on an example in which $N = 2048$ and $\alpha = .5$. The number of available processors P is varied, as is the dimension of the state vector n. Most of the entries in Tables 2 and 3 are relative times, which are calculated by dividing each absolute time by the *time required for serial algorithm* DP, *executed on a single processor of the system*, and multiplied by 100 percent.

Table 2 gives the theoretical optimal values for σ and τ, together with the time τ_{DP} required for the top level of the process, namely, execution of algorithm PDP

TABLE 2
Ncube: optimal number of stages and normalized runtimes for $N = 2048$, $\alpha = .5$, and various P and n. Runtimes expressed as percentages of runtime for Algorithm DP *on a single processor.*

P	n	σ_{opt}	$\tau_{\text{opt}} - \tau_{DP}$	τ_{DP}
16	3	3.59	.83	11.63
	5	4.28	.71	
	10	4.91	.65	
64	3	5.22	1.28	2.84
	5	6.35	1.08	
	10	7.36	0.99	

TABLE 3
Ncube: best feasible schedules and normalized runtimes for $N = 2048$, $\alpha = .5$, and various P and n. Runtimes expressed as percentages of runtime for Algorithm DP *on a single processor.*

P	n	σ	schedule	$\tau_\sigma - \tau_{DP}$
16	3	4	4,8,16	.84
	5	4	4,8,16	.71
	10	5	2,4,8,16	.67
64	3	5	4,8,32,64	1.34
	5	6	4,8,16,32,64	1.09
	10	6	4,8,16,32,64	1.00

on the P available processors. Here, $\tau_{\text{opt}} = \tau_{\sigma,\text{opt}}$ with $\sigma = \sigma_{\text{opt}}$. Since the ratio of communication cost to computation cost is not too high for this machine, the σ_{opt} values are quite close to the cyclic reduction maximum of $\sigma = \log_2 P + 1$. Table 3 gives the best feasible schedules for the various P and n; it can be observed that the times required are very close to the theoretical optima from Table 2.

Efficiency of a P-processor algorithm can be defined by the general formula

$$\text{efficiency} = T_S/(P * T_P),$$

where T_S is the execution time for the best serial algorithm on a single processor, and T_P is the execution time for the P-processor parallel algorithm. From Table 2, we see, for example, that when $n = 10$ and $P = 16$, the parallel algorithm requires 12.3 percent of the execution time of the serial method. Hence, $T_P/T_S = .123$, and the efficiency is .51. When $n = 10$ and $P = 64$, the efficiency is still .41. These are quite competitive with the efficiencies attained in the shared-memory case.

A recently released hypercube, based on the Intel i860 chip, is characterized by a very high ratio of communication cost to computation cost. Despite this, a similar timing analysis on the problem above with the i860 parameters in place of the Ncube parameters showed that efficiencies of .42 and .21 could be still attained on 16 and 128 processors, respectively, for the problems in Tables 2 and 3. These results are very encouraging, given the fine-grained nature of the latter calculation.

6. Computational results. Results of the implementation of the algorithms on shared-memory and distributed-memory machines are given here. We use two simple test problems with small state and control dimension, but with a large number

of stages N. Both are discretizations of continuous-time problems, with an Euler discretization being applied to the original state equation

$$\dot{x}(t) = f(x, u, t).$$

PROBLEM 1 (Bertsekas [2]). ($m = 1$, $n = 2$.) Choosing $h = 1/N$,

$$\begin{aligned} &\min\ h\textstyle\sum_{i=1}^{N} 6u_i^2 + 2x_{i+1,1}^2 + x_{i+1,2}^2 \\ \text{s.t.}\quad &x_{i+1} = \begin{bmatrix} 1 & h \\ -h & 1 \end{bmatrix} x_i + \begin{bmatrix} 0 \\ h \end{bmatrix} u_i, \qquad i = 1, \cdots, N. \end{aligned}$$

PROBLEM 2 (Russell [12]). ($m = 2$, $n = 3$.) $h = 1/N$,

$$\begin{aligned} &\min\ h\textstyle\sum_{i=1}^{N} 4u_{i,1}^2 + u_{i,2}^2 + x_{N+1}^T Q x_{N+1}, \\ &\text{s.t.}\quad x_{i+1} = (I + h\hat{A})x_i + h\hat{B}u_i, \end{aligned}$$

where

$$Q = \begin{bmatrix} 5.2478 & -5.2896 & 0 \\ -5.2896 & 7.5183 & -1.6938 \\ 0 & -1.6938 & 1.3119 \end{bmatrix},$$

$$\hat{A} = \begin{bmatrix} -2 & 2 & 0 \\ -.25 & -2.1706 & 1.8752 \\ 0 & -1 & -1 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} -.873 & 0 \\ 0 & -.873 \\ 0 & 0 \end{bmatrix}.$$

We also use both problems to test the constrained algorithms by imposing bounds on the components of u_i.

The Alliant FX/8, an eight-processor shared-memory machine, was used in the tests described below.

For the unconstrained problem, the following algorithms were implemented:

- Algorithm RI;
- Algorithm DP;
- The two-level algorithm consisting of PDP on eight processors, followed by RI to solve the reduced system;
- The three-level algorithm consisting of PDP on eight processors, followed by PRI on four processors and, finally, RI on a single processor.

The "serial" algorithms were compiled by using the `-Og` FORTRAN compiler option to run on a single processor, and also with the `-Ogc` option to run on all eight processors (and hence to reveal any parallelism in the algorithms). In the parallel version of RI, the initial elimination and final recovery of the state variables were done in a parallel loop. The two- and three-level parallel algorithms were also compiled to run both on a single processor and in concurrent mode. LINPACK routines were used to perform the various matrix factorizations and triangular solves.

Results are given in Table 4. Some improvement can be noted in the serial algorithms in concurrent mode, particularly for RI, because of the elementary optimization described above. The one-processor timings of the two- and three-level methods give an idea of how much "overhead" is involved in PDP, in the formation of matrices D_i, F_i, and $\tilde{J}_i$, and so on. It is seen that runtime increases by about 50 percent over serial DP. The eight-partition, eight-processor version of the two-level algorithms is about

TABLE 4
Alliant FX/8 runtimes (in seconds) for unconstrained Problems 1 and 2 ($N = 2000$).

	Problem 1		Problem 2	
	1 processor	8 processors	1 processor	8 processors
RI	3.15	1.90	6.56	3.25
DP	1.57	1.25	3.82	2.30
PDP—RI	2.48	.361	5.95	.820
PDP—PRI—RI	2.61	.379	6.27	.877

TABLE 5
Alliant FX/8 runtimes (in seconds) for gradient projection algorithm, Problem 1 ($N = 2000$).

	ubound=1.0 (18.4% active)		ubound=0.1 (81.5% active)	
	1 processor	8 processors	1 processor	8 processors
RI	11.8	7.94	17.8	10.1
DP	7.45	6.35	11.7	8.05
PDP—RI	12.2	2.18	14.2	2.76
PDP—PRI—RI	12.0	2.14	14.0	2.71

6.4 times faster than the eight-partition, one-processor version for the first problem and 7.1 times faster for the second problem. This speedup lags behind the ideal figure of 8, mainly because solution of the reduced system is a serial bottleneck. The three-level version of the algorithm took slightly longer, possibly because the overhead involved in additional levels of subroutine calls was not justified by the small amount of computation needed to solve the reduced system.

Defining "speedup" to be the ratio of the time taken by the best serial algorithm on one processor to the time taken for the best parallel algorithm on eight processors, we obtain a figure of 4.5 for the first problem and 4.6 for the second. These figures correspond well to the theoretical predictions of §5.

For the bound-constrained problem, the two-metric gradient projection framework from Bertsekas [2] is used, with each of the four unconstrained algorithms being used to solve the reduced system (47)–(50). The results are given in Tables 5 and 6. In the multiprocessor versions of the four codes, solution of the adjoint equation and evaluation of the states and objective function are parallelized as discussed in §4. The results are qualitatively similar to the unconstrained case, except that here the three-level algorithm has a slight advantage. Two factors inhibit perfect speedup in going from one to eight processors in the two- and three-level codes. The "serial" parts of these codes are proportionately more significant than in the unconstrained case, because of the small m and n values. There is also a load-balancing problem. An equal number of stages is assigned to each processor, but processing times for each partition vary because different numbers of u_i components are at their bounds within each partition. This means that the decrease factor in runtime in going from one to eight processors may be as low as 5.1 (as in Problem 1 with `ubound = 0.1`).

The number of iterations (i.e., solutions of reduced systems) is between 2 and 4 in each case. The sequence of iterates generated by each algorithm was the same, with the minor exception of the two serial methods, which because of roundoff error required an extra function evaluation of the last iteration of Problem 1 when the control bound was 0.1.

Speedups (ratio of best serial time to best parallel time) range from 3.5 to 4.3

TABLE 6
Alliant FX/8 runtimes (in seconds) for gradient projection algorithm, Problem 2 ($N = 2000$).

	ubound=0.2 (6.8% active)		ubound=0.05 (70.0% active)	
	1 processor	8 processors	1 processor	8 processors
RI	32.5	16.9	40.9	23.9
DP	21.4	14.3	27.8	17.8
PDP—RI	36.2	5.83	42.7	7.28
PDP—PRI—RI	35.3	5.63	42.0	7.13

and do not seem to depend strongly on the proportion of active constraints at the solution.

7. Continuous-time problems. The discrete-time problem and the algorithms discussed above have continuous-time analogs which we briefly describe in this section. These have an interesting relationship to known algorithms for two-point boundary value problems in ordinary differential and differential algebraic equations.

Given the problem (14), we can introduce the costate function q and apply standard constrained optimization techniques to obtain the following set of necessary conditions:

$$r(t) + R(t)^T y + S(t)v + B(t)^T q = 0, \tag{58}$$

$$\dot{q} + A(t)^T q + Q(t)y + R(t)v + z(t) = 0, \qquad q(T) = Q_f y(T) + z_f, \tag{59}$$

$$\dot{y} - A(t)y - B(t)v - s(t) = 0, \qquad y(0) = y_0. \tag{60}$$

This is a system of semi-explicit differential algebraic equations in q, v, and y; it will have index 1 if $S(t)$ is uniformly nonsingular on $[0, T]$. In fact, a standard second-order condition (the strengthened Legendre–Clebsch condition) for (14) to have a unique minimizer is that

$$S(t) \text{ is positive definite a.e. on } [0, T]. \tag{61}$$

It is, therefore, reasonable to use (58) to eliminate v from the system above and obtain the two-point boundary value problem

$$\dot{y} = \hat{A}(t)y + \hat{J}(t)q + \hat{s}(t), \qquad y(0) = y_0, \tag{62}$$

$$\dot{q} = -\hat{A}(t)^T q - \hat{Q}(t)y - \hat{z}(t), \qquad q(T) = Q_f y(T) + z_f, \tag{63}$$

where

$$\hat{A} = A - BS^{-1}R^T, \qquad \hat{J} = -BS^{-1}B^T, \qquad \hat{Q} = Q - RS^{-1}R^T,$$
$$\hat{s} = s - BS^{-1}r, \qquad \hat{z} = z - RS^{-1}r.$$

A continuous version of Algorithm RI can be deduced by making the Riccati substitution

$$q = K(t)y + b(t). \tag{64}$$

Combining (64) with (62)–(63), we obtain final value problems in K and b:

$$\dot{K} = -\hat{A}(t)^T K - K\hat{A}(t) - K\hat{J}(t)K - \hat{Q}(t), \qquad K(T) = Q_f, \tag{65}$$

$$\dot{b} = -[\hat{A}(t)^T - K(t)\hat{J}(t)]b - [K(t)\hat{s}(t) + \hat{z}(t)], \qquad b(T) = z_f. \tag{66}$$

Substitution into (62) then yields the following initial value problem for y:

$$\dot{y} = [\hat{A}(t) + \hat{J}(t)K(t)]y + [\hat{J}(t)b(t) + \hat{s}(t)], \qquad y(0) = y_0. \tag{67}$$

Hence it may be possible to solve (14) by integrating backward to solve (65) and (66), then integrating forward to solve (67), and finally obtaining q and v by substitution in (64) and (58), respectively. Of course, for this algorithm to work, we must be able to find a finite solution $K(t)$, $t \in [0,T]$, of (65). In the literature it is often simply assumed that such a solution exists (see, for example, Maurer [9]). However, Russell [12] and Polak [10] show that a finite $K(t)$ exists provided that, in addition to (61), we have

$$\begin{bmatrix} Q(t) & R(t) \\ R(t)^T & S(t) \end{bmatrix} \text{ is positive semidefinite a.e. on } [0,T]. \tag{68}$$

It is not difficult to derive an analog for Algorithm PRI. We first partition the interval $[0,T]$ into P subintervals using meshpoints $t_1, t_2, \cdots, t_{P+1}$ which satisfy

$$0 = t_1 < t_2 < \cdots < t_{P+1} = T.$$

Now we introduce the notation $\tilde{y}_i = y(t_i)$ and $\tilde{q}_i = q(t_i)$, $i = 1, \cdots, P+1$. The aim is now to express $y(t)$ and $q(t)$ purely in terms of $\tilde{y}_i$ and $\tilde{q}_i$, $i = 1, \cdots, P+1$, and to derive a "reduced" linear algebraic system in which the $\tilde{y}_i$ and $\tilde{q}_i$ are the unknowns. On each interval $[t_i, t_{i+1}]$ we make the "Riccati" substitution

$$q = K_i(t)y + b_i(t) + L_i(t)\tilde{q}_{i+1}, \qquad t \in [t_i, t_{i+1}]. \tag{69}$$

The usual manipulation yields final value problems for K_i, b_i, and L_i:

$$\dot{K}_i = -\hat{A}(t)^T K_i - K_i\hat{A}(t) - K_i\hat{J}(t)K_i - \hat{Q}(t), \qquad K_i(t_{i+1}) = 0, \tag{70}$$

$$\dot{b}_i = -[\hat{A}(t) + K_i(t)\hat{J}(t)]b_i - [K_i(t)\hat{s}(t) + \hat{z}(t)], \qquad b_i(t_{i+1}) = 0, \tag{71}$$

$$\dot{L}_i = -[\hat{A}(t)^T + K_i(t)\hat{J}(t)]L_i, \qquad L_i(t_{i+1}) = I. \tag{72}$$

By setting $\tilde{L}_i = L_i(t_i)$, $\tilde{K}_i = K_i(t_i)$, and $\tilde{b}_i = b_i(t_i)$, we deduce from (69) the following equation:

$$\tilde{q}_i = \tilde{K}_i\tilde{y}_i + \tilde{b}_i + \tilde{L}_i\tilde{q}_{i+1}.$$

To obtain another equation which relates the $\tilde{y}_i$ and $\tilde{q}_i$, we seek $H_i(t)$, $M_i(t)$, and $h_i(t)$ such that for $t \in [t_i, t_{i+1}]$,

$$H_i(t)y + M_i(t)\tilde{q}_{i+1} - \tilde{y}_{i+1} + h_i(t) = 0. \tag{73}$$

Substitution into (62) and (63) shows that

$$H_i(t) = L_i(t)^T, \tag{74}$$

and we obtain final-value problems in M_i and h_i:

$$\dot{M}_i = -L_i(t)^T\hat{J}(t)L_i(t), \qquad M_i(t_{i+1}) = 0, \tag{75}$$

$$\dot{h}_i = -L_i(t)^T[\hat{J}(t)b_i(t) + \hat{s}(t)], \qquad h_i(t_{i+1}) = 0. \tag{76}$$

By setting $t = t_i$ in (73) and using the notation $\tilde{M}_i = M_i(t_i)$, $\tilde{h}_i = h_i(t_i)$, we obtain

$$\tilde{L}_i^T\tilde{y}_i + \tilde{M}_i\tilde{q}_{i+1} - \tilde{y}_{i+1} + \tilde{h}_i = 0.$$

When the boundary conditions for $\tilde{y}_1$, $\tilde{q}_{P+1}$, and $\tilde{y}_{P+1}$ are included, we obtain a reduced linear system with unknown vector

$$(\tilde{q}_1, \tilde{y}_1, \tilde{q}_2, \tilde{y}_2, \cdots, \tilde{q}_{P+1}, \tilde{y}_{P+1}), \tag{77}$$

right-hand side

$$(-y_0^T, -\tilde{b}_1^T, -\tilde{h}_1^T, \cdots, -\tilde{h}_P^T, -z_f^T)^T, \tag{78}$$

and coefficient matrix

$$\begin{bmatrix} 0 & -I & & & & & & \\ -I & \tilde{K}_1 & \tilde{L}_1 & & & & & \\ & \tilde{L}_1^T & \tilde{M}_1 & -I & & & & \\ & & -I & \tilde{K}_2 & \tilde{L}_2 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & & \tilde{L}_P^T & \tilde{M}_P & -I \\ & & & & & & -I & Q_f \end{bmatrix}. \tag{79}$$

The partitioned algorithm can be summarized as follows: we first obtain $\tilde{K}_i$, $\tilde{b}_i$, $\tilde{L}_i$, $\tilde{M}_i$, and $\tilde{h}_i$, for $i = 1, \cdots, P$ by integration of (70), (71), (72), (75), and (76), respectively. Then, (77)–(79) is solved to find $\tilde{q}_i$, $\tilde{y}_i$, $i = 1, \cdots, P+1$. Finally, $y(t)$ and $q(t)$ can be recovered by substituting into (73) and (69).

The scheme just described can be placed in the framework of "theoretical multiple shooting" applied to the boundary value problem (62)–(63) (see Ascher and Mattheij [1]). We are effectively solving a set of P subproblems, one on each interval $[t_i, t_{i+1}]$, and for the ith subproblem we enforce the separated end conditions

$$q(t_{i+1}) = \tilde{q}_{i+1}, \qquad y(t_i) = \tilde{y}_i. \tag{80}$$

(At this point, of course, $\tilde{q}_{i+1}$ and $\tilde{y}_i$ are unknowns.) This approach contrasts with "standard" multiple shooting in which only initial conditions are enforced for each subproblem. We are effectively using Riccati substitution techniques to find solutions to the particular and fundamental problems

$$\begin{aligned} \dot{y}_{p,i} &= \hat{A}(t) y_{p,i} + \hat{J}(t) q_{p,i} + \hat{s}(t), & y_{p,i}(t_i) &= 0, \\ \dot{q}_{p,i} &= -\hat{A}(t)^T q_{p,i} - \hat{Q}(t) y_{p,i} - \hat{z}(t), & q_{p,i}(t_{i+1}) &= 0, \end{aligned}$$

$$\begin{aligned} \dot{Y}_{1,i} &= \hat{A}(t) Y_{1,i} + \hat{J}(t) P_{1,i}, & Y_{1,i}(t_i) &= 0, \\ \dot{P}_{1,i} &= -\hat{A}(t)^T P_{1,i} - \hat{Q}(t) Y_{1,i}, & P_{1,i}(t_{i+1}) &= I, \end{aligned}$$

$$\begin{aligned} \dot{Y}_{2,i} &= \hat{A}(t) Y_{2,i} + \hat{J}(t) P_{2,i}, & Y_{2,i}(t_i) &= I, \\ \dot{P}_{2,i} &= -\hat{A}(t)^T P_{2,i} - \hat{Q}(t) Y_{2,i}, & P_{2,i}(t_{i+1}) &= 0. \end{aligned}$$

It can be shown that the following identities hold:

$$\begin{aligned} Y_{1,i}(t_{i+1}) &= \tilde{M}_i, & Y_{2,i}(t_{i+1}) &= \tilde{L}_i^T, & y_{p,i}(t_{i+1}) &= \tilde{h}_i, \\ P_{1,i}(t_i) &= \tilde{L}_i, & P_{2,i}(t_i) &= \tilde{K}_i, & q_{p,i}(t_i) &= \tilde{b}_i. \end{aligned}$$

Given this connection to multiple shooting, we can briefly address the remaining issues associated with the partitioned method. One is the existence of solutions of the subproblems on each interval. Again, the conditions (61), (68) restricted to the interval $[t_i, t_{i+1}]$ are sufficient for the existence of a finite solution K_i to (70). Then, under weak assumptions on the coefficient functions in (14), we can deduce existence of solutions to (71), (72), (75), and (76).

A second important issue is the stability of this procedure. It is easy to see from (72) and (75) that there is a possibility of exponential growth in L_i (and hence M_i). A large body of literature has appeared in recent years on stability of numerical methods for two-point boundary value problems; from this we can state, loosely speaking, that if (62)–(63) is well conditioned (that is, not too sensitive to perturbations in the boundary conditions), then the method outlined above will be stable provided that the interval lengths are sufficiently small, and the reduced system (77)–(79) is solved in a stable way. The system (77)–(79) has the same form as the system (15)–(16) arising from Algorithm RI, and hence could be solved by the discrete algorithms RI or PRI. If such a procedure turns out to be unstable (something which is easily detected substituting the calculated solution into (77)–(79) and finding the residual), a stable parallel solver along the lines of those discussed in Wright [15] can be used instead.

Finally, we note that the partitioned algorithm just outlined, with its special choice of boundary conditions on each interval, is by no means the only possible parallel algorithm for (62)–(63), though it does seem to be a reasonably intuitive one. Another possibility is to use a global finite-differencing scheme to set up a large block-banded matrix, and then to use parallelism at the level of the linear algebra (again, see [15] for details).

8. Conclusions. We have described parallel methods for optimal control problems, and described their implementation on various parallel architectures. In the discrete-time case, the parallelism is implemented at the computationally intensive "inner loop" in which a linear-quadratic regulator problem (3) must be solved. This allows flexibility in the choice of nonlinear optimization framework to be used to solve the problem (1)–(2). Good speedups over the best serial algorithms were observed on a shared-memory machine, and some simple timing analysis shows that good results can also be expected on distributed-memory architectures. Under appropriate assumptions, the continuous-time "limit" of the partitioned algorithms is a special multiple shooting algorithm.

Acknowledgments. I thank the referees for their perceptive comments, which improved the manuscript.

REFERENCES

[1] U. M. Ascher and R. M. M. Mattheij, *General framework, stability and error analysis for numerical stiff boundary value problems*, Numer. Math., 54 (1988), pp. 355–372.

[2] D. P. Bertsekas, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221–246.

[3] S.-C. Chang, T.-S. Chang, and P. B. Luh, *A hierarchical decomposition for large-scale optimal control problems with parallel processing structure*, Automatica, 25 (1989), pp. 77–86.

[4] T. H. Dunigan, *Performance of the Intel i860 hypercube*, Tech. Report ORNL/TM-11491, Oak Ridge National Laboratory, Oak Ridge, TN, May 1990.

[5] J. C. DUNN AND D. P. BERTSEKAS, *Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems*, J. Optim. Theory Appl., 63 (1989), pp. 23–38.

[6] M. GAWANDE AND J. C. DUNN, *Variable metric gradient projection processes on convex feasible sets defined by nonlinear inequalities*, Appl. Math. Optim., 17 (1988), pp. 103–119.

[7] G. H. GOLUB AND C. F. VAN LOAN, *Matrix Computations*, Second Edition, The Johns Hopkins University Press, Baltimore, MD, 1989.

[8] C. T. KELLEY AND E. W. SACHS, *A pointwise quasi-Newton method for unconstrained optimal control problems*, Numer. Math., 55 (1989), pp. 159–176.

[9] H. MAURER, *First and second order sufficient optimality conditions in mathematical programming and optimal control*, Math. Programming Stud., 14 (1981), pp. 163–177.

[10] E. POLAK, *Computational Methods in Optimization*, Academic Press, New York, 1970.

[11] M. L. PSIAKI AND K. PARK, *Trajectory optimization for real-time guidance: Part 1, Time-varying* LQR *on a parallel processor*, Proc. 1990 American Control Conference, May 23–25, 1990, San Diego, CA, pp. 248–253.

[12] D. L. RUSSELL, *Mathematics of Finite-Dimensional Control Systems*, Marcel Dekker, New York, 1979.

[13] S. J. WRIGHT, *Parallel algorithms for banded linear systems*, SIAM J. Sci. Statist. Comput., to appear; Preprint MCS–P64–0289, Argonne National Laboratory, Argonne, IL, 1989.

[14] ———, *Solution of discrete-time optimal control problems on parallel computers*, Parallel Comput., 16 (1990), pp. 221–238.

[15] ———, *Stable parallel algorithms for two-point boundary value problems*, SIAM J. Sci. Statist. Comput., to appear; Preprint MCS–P178–0990, Argonne National Laboratory, Argonne, IL, 1990.

SIAM J. OPTIMIZATION
Vol. 1, No. 4, pp. 643–669, November 1991

013

ON THE FINE-GRAIN DECOMPOSITION OF MULTICOMMODITY TRANSPORTATION PROBLEMS*

STAVROS A. ZENIOS †

Abstract. A simple algorithm for nonlinear optimization problems with multicommodity transportation constraints is developed. The algorithm is of the row-action type and, when properly applied, decomposes the underlying graph alternatingly by nodes and arcs. Hence, a fine-grain decomposition scheme is developed that is suitable for massively parallel computer architectures of the SIMD (i.e., single instruction stream, multiple data stream) class.

Data structures are developed for the implementation of both dense and sparse problems, and details of an implementation on a Connection Machine CM–2 with 32K processing elements are given. The dense implementation achieves computing rate of up to three GFLOPS. Several aspects of the algorithm are investigated empirically. Computational results are reported for the solution of quadratic programs with approximately 10 million columns and 100 thousand rows.

Key words. multicommodity networks, nonlinear programming, massively parallel computing

AMS(MOS) subject classifications. 90C08, 90C30, 65K05

1. Introduction.

Data-level parallelism is a successful paradigm for computing on massively parallel architectures. It is based on the premise that a massively parallel algorithm should use multiple processors to carry out identical operations on different parts of the input data. Communication among processors is orderly and synchronous. With this approach one avoids the difficulties encountered in coordinating thousands—or, potentially, millions—of processors in an asynchronous, chaotic fashion. Even more important, however, is the ability to develop abstract models for data-level parallel computing: the vector random-access-machine (V-RAM) of Blelloch [5]. It is thus possible to study complexity issues of massively parallel algorithms, and gain insight into their efficiency in an abstract setting, even before actual implementations. A potential limitation of data-level parallelism is the requirement that the operations of the algorithm are identical on all the data. To what extent this mode of computing can be applied to solve a broad range of problems is presently unclear.

The focus of this paper is the design of *fine-grain decomposition* algorithms for nonlinear optimization problems with multicommodity transportation constraints. We seek algorithms that decompose the problem into a large number of independent and identical subproblems, and are, therefore, suitable for data-level parallel computing. If a problem requires $O(L)$ steps, a fine-grain decomposition will require $L \cdot O(1)$ steps. On a computer system with P processors the problem can be solved in $\lceil \frac{L}{P} \rceil \cdot O(1)$ steps. When P is large enough, as is the case with massively parallel computers, the problem can be solved in a constant number of operations which is independent of its size.

A fine-grain decomposition algorithm is designed here for a class of multicommodity transportation problems. Such problems appear in operations research (logistics, distribution, manufacturing, etc.), computer science (communication routing), and

* Received by the editors October 4, 1990; accepted for publication (in revised form) April 16, 1991. This research was supported in part by National Science Foundation grant SES–91-00216 and Air Force Office of Scientific Research grant 91-0168.

† Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, Pennsylvania 19104. This research was completed while the author was with Thinking Machines Corporation, Cambridge, Massachusetts 02142 and the Operations Research Center, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139.

transportation (the problem of estimating origin/destination tables, which is similar to the estimation of social accounting matrices in development planning and the estimation of migration patterns in regional sciences). This is the second in a series of three papers that develop massively parallel algorithms for optimization problems with network structures: The single commodity transportation problem was studied in Zenios and Censor [31], and the stochastic network problem in Nielsen and Zenios [24].

Linear multicommodity network flow problems have received extensive investigation; see the survey articles by Assad [1] or Kennington [21]. Very little has been done on the nonlinear problem, and in the cases known to the author it has been motivated by recent developments in parallel computing: Schultz and Meyer [27] and Pinar and Zenios [33]. The (easier) problem when the multiple commodities jointly contribute in a congestion function has been studied in greater length: Bertsekas [2] and Gallager [17] for data communication networks, and Chen and Meyer [14] for traffic assignment models. The algorithm considered here could also address a more general constraint set than these earlier studies: it permits multipliers (i.e., gains) on individual commodities, and weighted linear combinations of the commodities contribute to the coupling constraints. This is the *generalized* multicommodity flow problem. The only study that deals with this problem is Wollmer's [29].

The algorithmic approach we follow is based on the class of *row-action algorithms* of Censor [9]; see also Bregman [7], Censor and Lent [10], De Pierro and Iusem [25], and Elfving [16]. These algorithms have been proven successful in the solution of very large, sparse, optimization problems that arise in medical imaging (Herman [19]). With the appropriate choice of some control parameters, the algorithms can be implemented in parallel; for a classification see Censor [8], and for applications, see Zenios and Censor [32] and Censor and Zenios [12]. A loose interpretation of the algorithm is that it applies a Gauss–Seidel iteration to the first-order necessary optimality conditions. For each row of the constraint set the first-order system is solved by producing a pair of primal and dual variables that satisfy complementarity and are (primal) feasible for the specific row. Obviously, if two constraints do not share any primal variables, the two systems are independent, and if multiple processors are available they can be solved in parallel. As will be shown in this paper a parallel decomposition of the multicommodity transportation problem can be developed by iterating first over all the origin nodes for all commodities, then over the destination nodes for all commodities, and finally over the coupling constraints for each arc.

A limitation of our approach is that the developed algorithms are specialized for certain forms of objective functions (known as Bregman's functions). The quadratic and entropy functions are two special cases, and these functions are typically used in the matrix estimation applications mentioned above. However, these algorithms can not directly solve the linear programming case. They are, nevertheless, the building blocks for more general algorithmic schemes that can be applied to linear programs: the proximal minimization algorithm of Rockafellar [26], or the proximal minimization algorithm with D-functions (PMD) of Censor and Zenios [13].

The algorithm developed in this paper has been implemented on a massively parallel Connection Machine CM–2 with up to 32K processing elements. We develop the data structures for the representation of sparse, multicommodity network problems. To this end we extend the data structures of Zenios and Lasken [30] to represent multiple single commodity networks. The implementation is used to study the performance of the algorithm. Numerical results are analyzed for the solution of test

problems with up to 10 million variables and 100 thousand constraints.

We define now the problem and establish notation.

1.1. Problem formulation. We use $\langle m \rangle$ to denote the set $\{1, 2, 3, \cdots, m\}$. $\Re^m$ is the m-dimensional Euclidean space and $\langle \cdot, \cdot \rangle$ is the Euclidean inner product. $\text{mid}\{\alpha, \beta, \gamma\}$ is the median of the the three real numbers, α, β, and γ. If $\alpha \leq \gamma$ the median can be computed as $\max\{\alpha, \min\{\beta, \gamma\}\}$. We use $\lceil m \rceil_2$ to denote rounding up of m to the next integer that is a power of 2 (e.g., $\lceil 65 \rceil_2 = 128$).

A transportation graph is defined by the triplet $G = (V_O, V_D, \mathcal{E})$ where $V_O = \langle m_O \rangle$, $V_D = \langle m_D \rangle$ and $\mathcal{E} \subseteq \{(i,j) | i \in V_O, j \in V_D\}$. V_O and V_D are the sets of origin and destination nodes with cardinality m_O and m_D, respectively. $\mathcal{E}$ is the set of n directed arcs (i,j) with origin i and destination j which belong to the graph $(n \leq m_O\, m_D)$. On this transportation problem we consider the flow of K distinct commodities and let $\langle K \rangle$ denote the set of these commodities. We assume for symmetry of notation that the underlying graph is identical for all commodities. Additional notation is needed to define the transportation problem for each commodity, and then to define the joint restrictions over all the commodities. For each commodity $k \in \langle K \rangle$ we have:

$x_k = (x_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of flows,
$u_k = (u_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of upper bounds on the flows,
$s_k = (s_k(i)) \in \Re^{m_O}$, $i \in V_O$, is the vector of supplies,
$d_k = (d_k(j)) \in \Re^{m_D}$, $j \in V_D$, is the vector of demands,
$\pi_k^O = (\pi_k^O(i)) \in \Re^{m_O}$, $i \in V_O$, is the vector of dual prices for the origin nodes,
$\pi_k^D = (\pi_k^D(j)) \in \Re^{m_D}$, $j \in V_D$, is the vector of dual prices for the destination nodes,
$r_k = (r_k(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, is the vector of dual prices for the bound constraints.

For the joint capacity constraints we define:

$U = (U(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, the vector of mutual arc capacities,
$\psi = (\psi(i,j)) \in \Re^n$, $(i,j) \in \mathcal{E}$, the vector of dual prices for the joint capacity constraints.

In order to define the node-arc incidence relationships—assumed to be identical for all commodities—we define:

$\delta^+(i) = \{j \in V_D | (i,j) \in \mathcal{E}\}$, the set of destination nodes which have incident arcs with origin node i. We will use m_i to denote the cardinality of this set.
$\delta^-(j) = \{i \in V_O | (i,j) \in \mathcal{E}\}$, the set of origin nodes which have incident arcs with destination node j.

We define the pure multicommodity transportation problems as follows:

[PMTR] Pure Multicommodity Transportation Problem

(1) Minimize $F(x)$

Subject to :

(2) $$\sum_{j \in \delta^+(i)} x_k(i,j) = s_k(i), \quad \forall i \in V_O, \quad k \in \langle K \rangle$$

(3) $$\sum_{i \in \delta^-(j)} x_k(i,j) = d_k(j), \quad \forall j \in V_D, \quad k \in \langle K \rangle$$

(4) $$0 \leq x_k(i,j) \leq u_k(i,j), \quad \forall (i,j) \in \mathcal{E}, \quad k \in \langle K \rangle$$

(5) $$\sum_{k \in <K>} x_k(i,j) \leq U(i,j), \quad \forall (i,j) \in \mathcal{E}$$

The objective function $F : \Re^{nK} \to \Re$ is of the form:

[Q] Quadratic

$$F(x) = \sum_{k \in \langle K \rangle} \sum_{(i,j) \in \mathcal{E}} \frac{1}{2} w_k(i,j) \cdot x_k^2(i,j) + c_k(i,j) \cdot x_k(i,j) \tag{6}$$

where $w_k = (w_k(i,j))$ and $c_k = (c_k(i,j))$ are given vectors whose components are positive real numbers.

This discussion concludes the formulation of the problem for which we develop the algorithm. The following assumptions are made for the problem.

ASSUMPTION 1. The feasible set defined by (2)–(5) is nonempty. (The existence of feasible solutions for multicommodity network flow constraints can be checked using the algorithm in Gondran and Minoux [18, pp. 251–256].)

ASSUMPTION 2. The transportation graph G is connected. (Otherwise, the problem could be partitioned into its disconnected components and each solved separately.)

ASSUMPTION 3. $s_k(i) > 0$, for all $i \in V_O$, $k \in \langle K \rangle$ and $d_k(j) > 0$ for all $j \in V_D$, $k \in \langle K \rangle$. (If these conditions are violated for some index i or j for some commodity k, then all the flows for the particular commodity on arcs incident to the offending node can be set to zero, and the relevant constraint removed from the problem.)

1.2. Matrix formulations. In order to develop the row-action algorithms we will need to express the problems in a compact matrix notation. Constraints (2)–(5) are rewritten as:

$$Sx = s, \tag{7}$$

$$Dx = d, \tag{8}$$

$$0 \leq Ix \leq u, \tag{9}$$

$$0 \leq Ex \leq U. \tag{10}$$

I is the $nK \times nK$ identity matrix. $x \in \Re^{nK}$ is the vector $[x_1 \mid x_2 \mid \cdots \mid x_K]^T$, $u \in \Re^{nK}$ is the vector $[u_1 \mid u_2 \mid \cdots \mid u_K]^T$, $s \in \Re^{m_O K}$ is the vector $[s_1 \mid s_2 \mid \cdots \mid s_K]^T$, and $d \in \Re^{m_D K}$ is the vector $[d_1 \mid d_2 \mid \cdots \mid d_K]^T$.

S is the block-diagonal matrix:

$$S = \begin{bmatrix} S_1 & & & \\ & S_2 & & \\ & & \cdots\cdots & \\ & & & S_K \end{bmatrix}.$$

The blocks are defined as follows: for each commodity $k \in \langle K \rangle$ let $S_k = \hat{S}$ where $\hat{S}$ is the $m_O \times m_O m_D$ matrix:

$$\hat{S} = \left[\begin{array}{cccc|cccc|c|cccc} \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1m_D} & & & & & & & & \\ & & & & \alpha_{21} & \alpha_{22} & \cdots & \alpha_{2m_D} & \cdots & & & \\ & & & & & & & & & \alpha_{m_O 1} & \alpha_{m_O 2} & \cdots & \alpha_{m_O m_D} \end{array}\right].$$

The entries a_{ij} are given by

$$\alpha_{ij} = \begin{cases} 1, & \text{if } j \in \delta^+(i), \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

The remaining entries of $\hat{S}$ are all zeros.

D is the block-diagonal matrix:

$$D = \begin{bmatrix} D_1 & & & \\ & D_2 & & \\ & & \cdots\cdots & \\ & & & D_K \end{bmatrix}.$$

For each commodity $k \in \langle K \rangle$ we define D_k as the $m_D \times m_O m_D$ matrix:

$$D_k = \left[\begin{array}{ccc|ccc|c|ccc} \beta_{11} & & & \beta_{21} & & & & \beta_{m_O 1} & & \\ & \beta_{12} & & & \beta_{22} & & \cdots & & \beta_{m_O 2} & \\ & & \cdots & & & \cdots & & & & \cdots \\ & & \beta_{1m_D} & & & \beta_{2m_D} & & & & \beta_{m_O m_D} \end{array}\right].$$

with entries β_{ij} given by

$$\beta_{ij} = \begin{cases} 1, & \text{if } i \in \delta^-(j), \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

The remaining entries of D_k are all zeros. E is a *generalized upper bound* (GUB) constraint matrix of dimension $n \times nK$ of the form

$$E = [I_n \mid I_n \mid \cdots \mid I_n], \tag{13}$$

where I_n denotes the $n \times n$ identity matrix.

We also use

$$z = [\pi^O \mid \pi^D \mid r \mid \psi]^T$$

to denote the vector of dual prices, where $\pi^O \in \mathfrak{R}^{m_O K}$ is the vector $[\pi_1^O \mid \pi_2^O \mid \cdots \mid \pi_K^O]^T$, $\pi^D \in \mathfrak{R}^{m_D K}$ is the vector $[\pi_1^D \mid \pi_2^D \mid \cdots \mid \pi_K^D]^T$, and $r \in \mathfrak{R}^{nK}$ is the vector $[r_1 \mid r_2 \mid \cdots \mid r_K]^T$.

Finally, let

$$\Phi = [S \mid D \mid I \mid E]^T, \quad \gamma = [s \mid d \mid 0 \mid 0]^T, \quad \text{and} \quad \delta = [s \mid d \mid u \mid U]^T$$

and let $\phi^\ell = (\phi_t^\ell) = (\phi_{\ell t})$ denote the ℓth column of Φ^T, the transpose of Φ. We use lexicographic ordering for the variables, whereby the variable $x_k(i,j)$ is replaced by x_t with $t = (k-1)n + \sum_{l=0}^{i-1} m_l + j$. (Recall that $m_l = \text{card}(\delta^+(l))$ is the number of arcs emanating from node l. m_0 is taken to be equal to zero.) The row indices of Φ are partitioned into four sets:

$I_O = \{l \mid l = m_O(k-1) + i,\ \forall i \in \langle m_O \rangle,\ k \in \langle K \rangle\}$. This is the index set for rows of the equality constraints over all origin nodes (i.e., rows of (7)).

$I_D = \{l \mid l = m_O K + m_D(k-1) + j,\ \forall j \in \langle m_D \rangle,\ k \in \langle K \rangle\}$. This is the index set for rows of the equality constraints over all destination nodes (i.e., rows of (8)). The index set for rows corresponding to both origin and destination nodes is denoted by $I_1 = I_O \cup I_D$.

$I_2 = \{l = (m_O + m_D)K + q,\ \forall q \in \langle nK \rangle\}$, corresponding to the simple bounds of (9).

$I_3 = \{l = (m_O + m_D + n)K + q,\ \forall q \in \langle n \rangle\}$. This is the index set for rows of the GUB constraints (i.e., rows of (10)).

With this notation the pure multicommodity problem can be expressed as

$$\text{Minimize} \quad F(x) \tag{14}$$

$$\text{Subject to} \quad \gamma \leq \Phi x \leq \delta. \tag{15}$$

We will be developing the algorithms starting from the compact matrix notation, but will be expressing them finally in the algebraic representation of (2)–(5). It is only in the latter form that the fine-grain decomposition becomes apparent.

1.3. Outline of the paper. In §2 we develop the algorithm. The general row-action framework is first summarized. It is then used to develop the fine-grain decomposition algorithm for pure quadratic network problems, and discuss extensions to generalized networks. Section 3 develops the data structures for the implementation of the algorithm on massively parallel computers of the SIMD (i.e., single instruction stream, multiple data stream) class. Section 4 presents results from numerical experiments conducted on a Connection Machine CM–2 with up to 32K processing elements. The experiments establish the efficiency of the algorithm for the solution of very large problems. They also provide some insight on the performance of the algorithm for different problem characteristics. Section 5 presents the conclusions of our study and discusses directions for further research.

2. The fine-grain parallel algorithms. We begin with a sketch of the main idea. A fine-grain decomposition of the multicommodity transportation problem is developed when properly applying a row-action iterative algorithm. Such an algorithm operates on one row of the constraint set at a time—hence its name. The iterative step consists of an adjustment of the dual price of the constraint followed by an adjustment of the primal variables. Throughout, complementarity is preserved and upon completion of the algorithm primal feasibility is achieved. The algorithm iterates using an almost-cyclic control sequence over all the constraints. (The notion of almost-cyclic control was introduced in the analysis of Censor and Lent [10]. By definition, a sequence $\{i_k\}$ is almost-cyclic on the finite set I if $i_k \in I$ for all $k \geq 0$, and $I \subseteq \{i_{k+1}, \cdots, i_{k+C}\}$ for all $k \geq 0$ and some integer C. The constant C is referred to as the constant of almost-cyclicality.)

Obviously, if two constraints do not share any primal variables, the respective iterative steps can be executed independently. If multiple processors are available they can also be executed in parallel. It is then desirable to use the almost-cyclic control mechanism to choose independent constraints. In the case of the multicommodity transportation problem the algorithm first iterates on the origin nodes for all the commodities, then it iterates on the destination nodes for all the commodities, then it iterates on the simple bounds for all the commodities, and finally it iterates on the GUB constraints. One iteration of the algorithm requires $(m_O + m_D + n)K + n$ operations. When iterating on the origin nodes it requires $m_O K$ operations, on the destination nodes it requires $m_D K$ operations, on the simple bounds it requires nK operations, and on the joint capacity constraints it requires n operations. Hence, if the number of processors P scales linearly with $\max\{nK,\ m_O K,\ m_D K\}$, the algorithm can be executed in a constant number of operations per iteration. The rest of this section makes these ideas precise.

2.1. The row-action framework. We need some preliminary discussion. Let $F : \Lambda \subseteq \Re^n \to \Re$ where Λ is an arbitrary nonempty set, and let $S \neq \phi$ be an open

convex set such that its closure $\bar{S} \subseteq \Lambda$. The set S is called the *zone* of F, if F is strictly convex and continuous on $\bar{S}$ and continuously differentiable on S.

Let $D(x,y) = F(x) - F(y) - \langle \nabla F(y), x - y \rangle$, and let $H(a,b)$ be the hyperplane $H(a,b) = \{x \in \Re^n \mid \langle a, x \rangle = b\}$. The D-projection (or Bregman projection) of a point y onto $H(a,b)$ is defined by

$$P_{H(a,b)}(y) = \arg \min_{x \in H(a,b) \cap \bar{S}} D(x,y). \tag{16}$$

A function F—that belongs to the family of *Bregman's functions* as characterized by Censor and Lent [10]—has the zone consistency property with respect to the hyperplane $H(a,b)$ if the D-projection of every $y \in S$ onto $H(a,b)$ is also in S. If a function is zone consistent with respect to $H(a,b)$, then it can be shown (Censor and Lent [10, Lemma 3.1]) that the D-projection of y onto $H(a,b)$ is the point x given by the unique solution of the system of equations in x and β:

$$\nabla F(x) = \nabla F(y) + \beta \cdot a, \tag{17}$$

$$\langle a, x \rangle = b. \tag{18}$$

The unique real number β is known as the *Bregman* parameter. It can be interpreted as a step on the dual price for equation $\langle a, x \rangle = b$ that, together with the primal variable x, preserves the complementary slackness condition (17) while x satisfies equation (18).

The quadratic function [Q] is a Bregman's function (Censor and Lent [10]) with zone $\Re^n$. It is also easy to verify that, under Assumptions 1–3, it has the strong zone consistency property with respect to the hyperplanes $H(\phi^\ell, \gamma_\ell)$ and $H(\phi^\ell, \delta_\ell)$ for all $\ell \in I_1 \cup I_2 \cup I_3$. Hence we can apply the following general iterative scheme.

Algorithm 2.1. General Row-Action Algorithm for Mixed Equality and Inequality Constraints.

Step 0: (Initialization.) $\nu \leftarrow 0$. Compute $z^0 \in S$ and $x^0 \in \Re^n$ such that

$$\nabla F(x^0) = -\Phi^T z^0. \tag{19}$$

Step 1: (Iterative step over equality constraints.) Choose a row index $\ell(\nu) \in I_1$, and solve the following system for x^ν and β^ν:

$$\nabla F(x^{\nu+1/2}) = \nabla F(x^\nu) + \beta_\nu \phi^{\ell(\nu)}, \tag{20}$$

$$z^{\nu+1/2} = z^\nu - \beta_\nu e^{\ell(\nu)}, \tag{21}$$

where β_ν is the Bregman parameter associated with the D-projection of x^ν on the hyperplane $H(\phi^{\ell(\nu)}, \gamma_{\ell(\nu)})$. $\{\ell(\nu)\}$ is the control sequence of the algorithm, henceforth abbreviated as $\ell = \ell(\nu)$. $e^\ell \in \Re^{m_O + m_D + Kn + n}$ is the ℓth standard basis vector having 1 in the ℓth coordinate and zeros elsewhere.

Step 2: (Iterative step over interval constraints.) Choose a row index $\ell(\nu) \in I_2 \cup I_3$, and calculate Γ_ν and Δ_ν where Γ_ν and Δ_ν are the Bregman parameters associated with the D-projection of $x^{\nu+1/2}$ on the hyperplanes specified when the left and right inequalities of (15), respectively, of the interval constraints hold with equality (i.e., $x^{\nu+1/2}$ is projected on the hyperplanes $H(\phi^{\ell(\nu)}, \gamma_{\ell(\nu)})$ and $H(\phi^{\ell(\nu)}, \delta_{\ell(\nu)})$, respectively). Hence, compute $x^{\nu+1}$ and $z^{\nu+1}$ as follows:

$$\beta_\nu = \text{mid}\{z_{\ell(\nu)}^{\nu+1/2},\ \Gamma_\nu,\ \Delta_\nu\}, \tag{22}$$

$$\nabla F(x^{\nu+1}) = \nabla F(x^{\nu+1/2}) + \beta_\nu \phi^{\ell(\nu)}, \tag{23}$$

$$z^{\nu+1} = z^{\nu+1/2} - \beta_\nu e^{\ell(\nu)}. \tag{24}$$

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.2. Algorithm for quadratic problems. We now specialize the general row-action scheme for quadratic programs with pure multicommodity network constraints. The iterative step for the equality constraints is derived from Algorithm 2.1 (Step 1). It takes the form:

$$x_t^{\nu+1} = x_t^\nu + \frac{\beta_\nu}{w_t}\phi_t^\ell, \qquad t = 1, 2, \cdots, Kn, \tag{25}$$

$$z^{\nu+1} = z^\nu - \beta_\nu e^\ell, \tag{26}$$

where $\ell \in I_1$. The parameter β_ν is obtained by solving

$$y_t = x_t^\nu + \frac{\beta_\nu}{w_t}\phi_t^\ell, \qquad t = 1, 2, \cdots, Kn, \tag{27}$$

$$\langle \phi^\ell, y \rangle = \gamma_\ell. \tag{28}$$

For any control index $\ell \in I_O$ the iterative step is obtained as follows: First, by the definition of I_O, ℓ can be expressed as $\ell = m_O(k-1) + i$ for some $i \in \langle m_O \rangle$ and $k \in \langle K \rangle$. Hence, ϕ^ℓ is the ith row of the kth block of matrix S with entries α_{ij} as given by (11). x_t^ν will be updated according to (25) only when $\phi_t^\ell \neq 0$. This occurs for values of t that correspond to the lexicographic ordering of variable $x_k(i,j)$ for $j \in \delta^+(i)$, since it is only for these values that $\alpha_{ij} = 1$. Furthermore, the ℓth component of γ is $s_k(i)$ with dual variable $\pi_O^k(i)$. Hence, system (27)–(28) can be simplified to

$$y_k(i,j) = x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \qquad j \in \delta^+(i), \tag{29}$$

$$\sum_{j \in \delta^+(i)} y_k(i,j) = s_k(i), \tag{30}$$

for all $k \in \langle K \rangle$. Solving this system for β_ν we obtain

$$\beta_\nu = \frac{1}{\sum_{j \in \delta^+(i)} \frac{1}{w_k(i,j)}} \left[s_k(i) - \sum_{j \in \delta^+(i)} x_k^\nu(i,j) \right]. \tag{31}$$

This expression for β_ν is substituted in (25)–(26) to complete the iterative step. Similarly, we obtain the iterative step for any control index $\ell \in I_D$.

The iterative step for the simple bound constraints (i.e., $\ell \in I_2$) can be obtained as a simplification of the interval-constrained step. It is identical to the step for the single commodity quadratic transportation problems (see Zenios and Censor [30]) and its derivation is not repeated here.

For any control index $\ell \in I_3$ the iterative step is taken over the interval constraints. It is obtained from Algorithm 2.1 (Step 2):

$$x_t^{\nu+1} = x_t^{\nu} + \frac{\beta^{\nu}}{w_t}\phi_t^{\ell}, \qquad t = 1, 2, \cdots, Kn, \tag{32}$$

$$z^{\nu+1} = z^{\nu} - \beta_{\nu} e^{\ell}, \tag{33}$$

where $\beta_{\nu} = \text{mid}\{z_{\ell}^{\nu}, \Gamma_{\nu}, \Delta_{\nu}\}$. Γ_{ν} is obtained by solving

$$y_t = x_t^{\nu} + \frac{\Gamma_{\nu}}{w_t}\phi_t^{\ell}, \qquad t = 1, 2, \cdots, Kn, \tag{34}$$

$$\langle \phi^{\ell}, y \rangle = \gamma_{\ell}, \tag{35}$$

and Δ_{ν} is obtained by solving

$$y_t = x_t^{\nu} + \frac{\Delta_{\nu}}{w_t}\phi_t^{\ell}, \qquad t = 1, 2, \cdots, Kn, \tag{36}$$

$$\langle \phi^{\ell}, y \rangle = \delta_{\ell}. \tag{37}$$

In order to solve for Γ_{ν} or Δ_{ν}, we need once more to examine the structure of ϕ^{ℓ} when $\ell \in I_3$, i.e., $\ell = (m_O + m_D + n)K + q$ for some $q \in \langle n \rangle$. ϕ^{ℓ} is the qth row of the matrix E with entries e_{ij}, as given by (13). x_t^{ν} will be updated according to (32) only when $\phi_t^{\ell} \neq 0$. This occurs for values of t that correspond to the lexicographic ordering of variables $x_k(i,j)$ that satisfy $t = (k-1)n + q$ for all $k \in \langle K \rangle$ (i.e., only variables with lexicographic order $q, n+q, 2n+q, \cdots, (K-1)n+q$ will appear in the qth GUB constraint). Furthermore, the ℓth component of γ is 0, the ℓth component of δ is $U(i,j)$, and the dual variable is $\psi(i,j)$. With these observations, systems (34)–(35) and (36)–(37) can be simplified to:

$$y_k(i,j) = x_k^{\nu}(i,j) + \frac{\Gamma_{\nu}}{w_k(i,j)}, \qquad k \in \langle K \rangle, \tag{38}$$

$$\sum_{k=1}^{K} y_{k(i,j)} = 0, \tag{39}$$

and

$$y_k(i,j) = x_t^{\nu}(i,j) + \frac{\Delta_{\nu}}{w_k(i,j)}, \qquad k \in \langle K \rangle \tag{40}$$

$$\sum_{k=1}^{K} y_k(i,j) = U(i,j). \tag{41}$$

Solving these systems, we obtain

$$\Gamma_{\nu} = -\frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}} \sum_{k=1}^{K} x_k^{\nu}(i,j), \tag{42}$$

and

$$\Delta_\nu = \frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}} \left[U(i,j) - \sum_{k=1}^{K} x_k^\nu(i,j) \right]. \tag{43}$$

These expressions for Γ_ν and Δ_ν are used to compute β_ν using the mid{ } operator, i.e., (22), which is then substituted in (32)–(33) to complete the iterative step.

We now have all the components required to complete the algorithm for pure multicommodity transportation problems with a quadratic objective function.

Algorithm 2.2. Quadratic Optimization Algorithm for Pure Multicommodity Transportation Problems.

Step 0: (Initialization) $\nu \leftarrow 0$. $z^0 \leftarrow 0$,

$$x_k^0(i,j) = -\frac{c_k(i,j)}{w_k(i,j)}, \quad \forall k \in \langle K \rangle, \quad (i,j) \in \mathcal{E}. \tag{44}$$

Step 1: (Solve the single commodity problems.)
FOR $k = 1, 2, 3, \cdots, K$:
Step 1.1: (Solve for origin nodes.)
FOR $i = 1, 2, 3, \cdots, m_O$:
Compute

$$\beta_\nu = \frac{1}{\sum_{j \in \delta^+(i)} \frac{1}{w_k(i,j)}} \left[s_k(i) - \sum_{j \in \delta^+(i)} x_k^\nu(i,j) \right] \tag{45}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \qquad j \in \delta^+(i), \tag{46}$$

$$(\pi_k^O(i))^{\nu+1} = (\pi_k^O(i))^\nu - \beta_\nu \tag{47}$$

ENDFOR
Step 1.2: (Solve for destination nodes.)
FOR $j = 1, 2, 3, \cdots, m_D$:
Compute

$$\beta_\nu = \frac{1}{\sum_{i \in \delta^-(j)} \frac{1}{w_k(i,j)}} \left[d_k(j) - \sum_{i \in \delta^-(j)} x_k^\nu(i,j) \right] \tag{48}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \qquad i \in \delta^-(j), \tag{49}$$

$$(\pi_k^D(j))^{\nu+1} = (\pi_k^D(j))^\nu - \beta_\nu \tag{50}$$

ENDFOR
Step 1.3: (Solve for the simple bounds.)
FOR $(i,j) \in \mathcal{E}$:
Compute

$$\beta_\nu = \text{mid}\{r_k^\nu(i,j),\ w_k(i,j)\cdot(u_k(i,j)-x_k^\nu(i,j)),\ -w_k(i,j)\cdot x_k^\nu(i,j)\} \tag{51}$$

Update

$$x_k^\nu(i,j) \leftarrow x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \tag{52}$$

$$r_k^{\nu+1}(i,j) = r_k^\nu(i,j) - \beta_\nu \tag{53}$$

ENDFOR

ENDFOR

Step 2: (Solve for the joint capacity constraints.)

FOR $(i,j) \in \mathcal{E}$:

Compute

$$\Gamma_\nu = -\frac{1}{\sum_{k=1}^K \frac{1}{w_k(i,j)}} \sum_{k=1}^K x_k^\nu(i,j) \tag{54}$$

$$\Delta_\nu = \frac{1}{\sum_{k=1}^K \frac{1}{w_k(i,j)}} \left[U(i,j) - \sum_{k=1}^K x_k^\nu(i,j)\right] \tag{55}$$

$$\beta_\nu = \text{mid}\{\psi^\nu(i,j), \Gamma_\nu, \Delta_\nu\} \tag{56}$$

Update

$$x_k^{\nu+1}(i,j) = x_k^\nu(i,j) + \frac{\beta_\nu}{w_k(i,j)}, \quad \forall\, k \in \langle K\rangle, \tag{57}$$

$$\psi^{\nu+1}(i,j) = \psi^\nu(i,j) - \beta_\nu \quad \forall\, k \in \langle K\rangle. \tag{58}$$

ENDFOR

Step 3: Let $\nu \leftarrow \nu + 1$, and return to Step 1.

2.3. Discussion.

2.3.1. Extensions to generalized networks. The algorithm of the previous section can be extended to solve generalized network problems, whereby the coefficients β_{ij} are not restricted to the values $\{0,1\}$, but may be arbitrary real numbers. The numerical values of these coefficients may also depend on the commodity k, in which case the coefficients of D_k will be of the form β_{ij}^k. As in the case of the pure network problem it is possible to obtain closed form solutions for the Bregman parameters.

Row-action algorithms can also be developed for the optimization of entropy functions of the form

$$F(x) = \sum_{k\in<K>} \sum_{(i,j)\in\mathcal{E}} x_k(i,j)\cdot\left[\ln\left(\frac{x_k(i,j)}{a_k(i,j)}\right) - 1\right],$$

where ln is the natural logarithm and $a_k(i,j)$ are given positive real numbers. For pure problems we obtain once more closed form solutions for the Bregman parameters. However, for the generalized network problem, the Bregman parameters are given as the solution of a nonlinear equation. It turns out that an approximate solution can be obtained in closed form without destroying the asymptotic convergence of the algorithm; refer to the convergence analysis of Censor et al. [11]. The development and testing of algorithms for the generalized network problem are the topic of current research and will be reported elsewhere. A sketch of the algorithms is given in the working paper version of this article.

2.3.2. Relaxation parameters. Relaxation parameters $\{\lambda_\nu\}$ can be built into the algorithm. Different relaxation parameters can be used for different constraints, and they can be changed as the algorithm proceeds, provided they are within some bounds imposed to guarantee convergence. For the quadratic algorithm, for example, $0 < \epsilon \leq \lambda_\nu \leq 2 - \epsilon$. The use of relaxation parameters could accelerate convergence of the algorithm to an approximate solution. Since the algorithms are usually terminated when a sufficiently "good" solution is obtained, the use of relaxation parameters could be of great practical significance. This topic deserves further study.

2.3.3. Choice of control sequence. The algorithms developed here use the following control sequence in choosing rows of the constraint set on which to compute the Bregman projections: (1) origin node constraints, (2) destination node constraints, (3) simple bounds, (4) GUB constraints. Of course, any other almost cyclic control sequence will do. It is unclear which control sequence would accelerate the convergence of the algorithm towards an approximate solution. In addition to empirical studies, it is possible to undertake a more fundamental analysis: The algorithms project the current iterate x^ν on successive hyperplanes that are specified by the choice of $\phi^{\ell(\nu)}$. If $\phi^{\ell(\nu)}$ and $\phi^{\ell(\nu+1)}$ are almost parallel the algorithm will take very small steps. Looking at the cosine of the angle between successive hyperplanes

$$\frac{\langle \phi^{\ell(\nu)}, \phi^{\ell(\nu+1)} \rangle}{\| \phi^{\ell(\nu)} \| \cdot \| \phi^{\ell(\nu+1)} \|}, \tag{59}$$

we can choose hyperplanes that are (almost) orthogonal. On the other hand, if some hyperplanes $\{\phi^\ell\}$ are almost parallel, they could be replaced by a surrogate hyperplane. Given the rich structure of the constraint matrix Φ, it is worthwhile to investigate specialized acceleration schemes, starting from the discussion of Björck and Elfving [4] or Bramley and Sameh [6].

2.3.4. Potentially difficult problems. It is possible to gain insight into the performance of the algorithm on a candidate class of test problems by examining the problem data and the structure of the algorithms. Refer, for example, to the quadratic programming Algorithm 2.2. In the dual step calculation (Step 1.1), we see the term

$$\frac{1}{\sum_{j \in \delta^+(i)} \frac{1}{w_k(i,j)}}.$$

If the coefficients $w_k(i,j)$ are very small, so will be the whole term and the algorithm will be taking very small steps. Similar performance will be observed for very large and dense problems.

2.3.5. Asymptotic convergence. Since the algorithm we developed is a specialization of the general row-action framework, its asymptotic convergence can be derived from known results. For pure problems and generalized quadratic problems, we can obtain convergence from the results of Censor and Lent [10] and Elfving's [16] extension to mixed equality and inequality constraints. For the relaxed versions of the algorithm, one needs to extend the results of De Pierro and Iusem [25] and Censor et al. [11] to the mixed equality and interval-constrained problem.

3. Massively parallel implementations. The motivation for the design of the algorithms has been the desire to exploit massively parallel computing for the solution of very large problems. Of particular interest is the concept of data-level parallelism, whereby the problem is decomposed into fine-grain identical operations executed on

multiple data. If a large number of processors is available, then each one could execute these operations on its local data elements.

When there is interaction among the problem data, it would be necessary to communicate among the corresponding processors. Such communication can be combined with computations (as, for example, when P processors with local data α_i, $i = 1, 2, 3, \cdots, P$, coordinate to compute the partial sums $\alpha_j = \sum_{\ell=1}^{j} a_\ell$) or the communication step could be void of any computing, as in the case of permuting the data among processors according to some index list $\alpha_i \leftarrow \alpha_{list(i)}$.

The algorithm we developed decomposes naturally for this form of parallelism. In this section we discuss implementations of the quadratic programming algorithm on a massively parallel Connection Machine CM–2.

3.1. The Connection Machine CM–2. We briefly introduce the characteristics of the Connection Machine (model CM–2) (Hillis [20]) that are relevant to the parallel implementations discussed here. The Connection Machine is a fine-grain SIMD (i.e., single instruction stream, multiple data stream) system. Its basic hardware component is an integrated circuit with 16 processing elements (PEs) and a *router* that handles general communication. A fully configured CM has 4,096 chips for a total of 65,536 PEs. The 4,096 chips are interconnected as a 12-dimensional hypercube. Each processor is equipped with local memory of 8Kbytes, and for each cluster of 32 PEs a floating-point accelerator handles floating-point arithmetic.

Operations by the PEs are under the control of a *microcontroller* that broadcasts instructions from a front-end computer (FE) simultaneously to all the elements for execution. A flag register at every PE allows for no-operations; i.e., an instruction received from the microcontroller is executed if the flag is set, and ignored otherwise.

Parallel computations on the CM are in the form of a single operation executed on multiple copies of the problem data. All processors execute identical operations, each one operating on data stored in its local memory, accessing data residing in the memory of other PEs, or receiving data from the front end. This mode of computation is termed *data level parallelism* in contradistinction to *control level parallelism*, whereby multiple processors execute their own control sequence, operating either on local or shared data.

To achieve high performance with data-level parallelism one needs a large number of processors. The CM provides the mechanism of *virtual processors* (VPs) that allows one PE to operate in a serial fashion on multiple copies of data. VPs are specified by slicing the local memory of each PE into equal segments and allowing the physical processor to loop over all slices. The number of segments is called the VP *ratio* (i.e., ratio of virtual to physical PEs). Looping by the PE over all the memory slices is executed, in the worst case, in linear time. The set of virtual processors associated with each element of a data set is called a VP *set*.

The CM supports two addressing mechanisms for communication. The *send* address is used for general purpose communications via the routers. The NEWS address describes the position of a VP in an n-dimensional grid that optimizes communication performance. The *send* address indicates the location of the PE (hypercube address) that supports a specific VP and the relative address of the VP in the VP set that is currently active. NEWS address is an n-tuple of coordinates that specifies the relative position of a VP in an n-dimensional Cartesian-grid geometry. A *geometry* is an abstract description of such an n-dimensional grid. Once a geometry is associated with the currently active VP set a relative addressing mechanism is established among the processors in the VP set. Each processor has a relative position in the n-dimensional

geometry and NEWS allows the communication across the North, East, West, and South neighbors of each processor, and enables the execution of operations along the axes of the geometry. Such operations are efficient since the n-dimensional geometry can be mapped onto the underlying hypercube in such a way that adjacent VPs are mapped onto vertices of the hypercube connected with a direct link. This mapping of an n-dimensional mesh on a hypercube is achieved through a Gray coding; see, e.g., Bertsekas and Tsitsiklis [3, p. 50].

The algorithm was implemented using C/Paris. Paris is a low level protocol by which the actions of the data processors of the CM are controlled by the front end. Before invoking Paris instructions from a program the user has to specify the VP set, create a geometry, and associate the VP set with the geometry. Thus a communications mechanism is established (along both *send* and NEWS addresses). Paris instructions—parallel primitives—can then be invoked to execute operations along some axis of the geometry (using NEWS addresses), operate on an individual processor using *send* addresses, or translate NEWS to *send* addresses for general interprocessor communication or communication with the front end.

Parallel primitives that are relevant to our implementation are the *scans* and *spreads* of Blelloch [5]. The $\otimes$-scan primitive, for an associative, binary operator $\otimes$, takes a sequence $\{x_0, x_1, \cdots, x_n\}$ and produces another sequence $\{y_0, y_1, \cdots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_i$. For example, *add-scan* takes as an argument a parallel variable (i.e., a variable with its ith element residing in a memory field of the ith VP) and returns at VP i, the value of the parallel variable summed over $j = 0, \cdots, i$. User options allow the scan to apply only to preceding processors (e.g., sum over $j = 0, \cdots, i-1$) or to perform the scan in reverse. The $\otimes$-spread primitive, for an associative binary operator $\otimes$, takes a sequence $\{x_0, x_1, \cdots, x_n\}$ and produces another sequence $\{y_0, y_1, \cdots, y_n\}$ such that $y_i = x_0 \otimes x_1 \otimes \cdots \otimes x_n$. For example, *add-spread* takes as an argument a parallel variable residing in the memories of n active data processors and returns at VP i, the value of the parallel variable summed over $j = 0, \cdots, n$.

Another variation of the scan primitives allows their operation within *segments* of a parallel variable. These primitives are denoted as *segmented-$\otimes$-scan.* They take as arguments a parallel variable and a set of segment bits which specify a partitioning of the VP set into contiguous segments. Segment bits have a "1" at the starting location of a new segment and "0" elsewhere. A *segmented-$\otimes$-scan* operation restarts at the beginning of every segment. When processors are configured as a NEWS grid, scans within rows or columns of the grid are special cases of segmented scans called *grid-scans.*

3.2. Dense implementation. In all implementations we assume that the individual commodities are unbounded from above (i.e., $u_k(i,j) = +\infty$, for all arcs and all commodities), and are only restricted through the GUB constraints. This is a reasonable practical assumption, and our implementations can be easily modified to remove this restriction. In the dense implementation of the algorithm, it is also assumed that the graph G is complete. The CM–2 is configured into a two-dimensional communication mesh, i.e., a NEWS grid, of dimensions $\lceil m_O \rceil_2 \times \lceil m_D \rceil_2$. This grid is then used to solve a sequence of single-commodity transportation problems, implemented as in Zenios and Censor [30]. The memory of VP with NEWS coordinates (i,j) stores the data for arc $(i,j) \in \mathcal{E}$. It is partitioned into the following data fields:

1. Supply and demand, $s = s_k(i)$ and $d = d_k(j)$;
2. Current iterate, $x = x_k(i,j)$;

3. Dual price PSI=$\psi(i,j)$;
4. Sum of the flows of all commodities ex=$\sum_{k=1}^{K} x_k(i,j)$;
5. Joint capacity constraint $U = U(i,j)$;
6. Three fields IW, JW, and KW that hold the constants

$$\frac{1}{\sum_{i\in\delta^-(j)} \frac{1}{w_k(i,j)}}, \quad \frac{1}{\sum_{j\in\delta^+(i)} \frac{1}{w_k(i,j)}}, \quad \text{and} \quad \frac{1}{\sum_{k=1}^{K} \frac{1}{w_k(i,j)}},$$

respectively, and a field W that holds the quadratic coefficient $w_k(i,j)$;
7. Scaling factor BETA that holds β_ν;
8. Scratch fields to hold intermediate results.

In the memory of the FE we define a vector of K structures of the form

```
a-commodity {
fe-w[m_O][m_D]
fe-x[m_O][m_D]
fe-s[m_O]
fe-d[m_D] } commodity-t;
commodity-t com[K] ;
```

Figure 1 illustrates the memory configuration of both the CM and the FE. With this layout of memory, Algorithm 2.2 is executed as follows:

Step 0: Initialize according to Step 0 and set BETA=0.

Step 1: Initialize ex to zero. Move one commodity at a time from the FE to the CM. Update x and PSI according to (57) and (58), and solve by executing iteratively Steps 1.1–1.3 until some local termination criteria are satisfied. Accumulate the optimal solution into field ex, and move the optimal solution from the CM to the FE.

Step 2: Compute the scaling factor from Step 2, equation (56), store in BETA, and return to Step 1.

3.3. Sparse implementation. When solving sparse network optimization problems on the CM we have to map the network topology to the virtual processors in a way that is efficient for both computations and communications. The data structures introduced in Zenios and Lasken [29] are at present the best known method to represent sparse network problems. A comparison of alternative parallel implementations is reported in Nielsen and Zenios [23], and these data structures have been used by Eckstein [15] for the implementation of his alternating directions method of multipliers, with very encouraging results. The representation adopted in these studies uses a one-dimensional geometry of size $\lceil 2n + (m_O + m_D)\rceil_2$. It assigns two VPs for each arc (i,j), one at the tail node i, and one at the head node j and one VP for each node. VPs that correspond to the same node are grouped together into a contiguous segment. In this way segmented-scan operations can be used for computing and for communicating data among processors incident to a node. The general communication of prices among nodes is a one-to-one send operation between the VPs at the head and tail of each arc.

In order to implement a sparse, multicommodity network solver, we use the single commodity nonlinear network optimizer of Nielsen and Zenios [28]. The single commodity solver can be easily extended to solve multiple independent commodities in parallel: The CM is configured as a two-dimensional communications grid of dimensions $\lceil K\rceil_2 \times \lceil 2n + (m_O + m_D)\rceil_2$. Each row of the 0-axis is used to represent a single network problem as outlined above. Since the network problem has identical

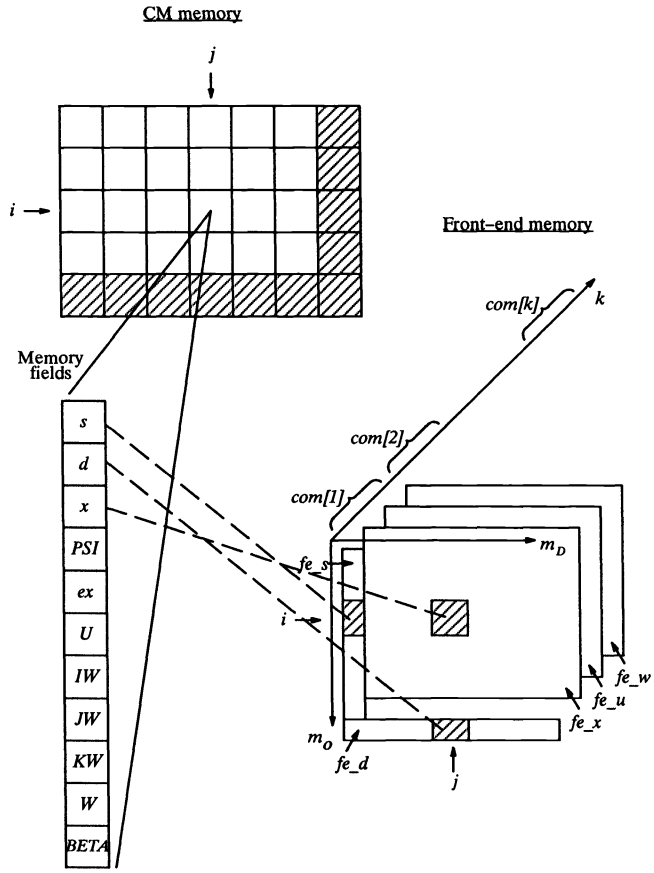
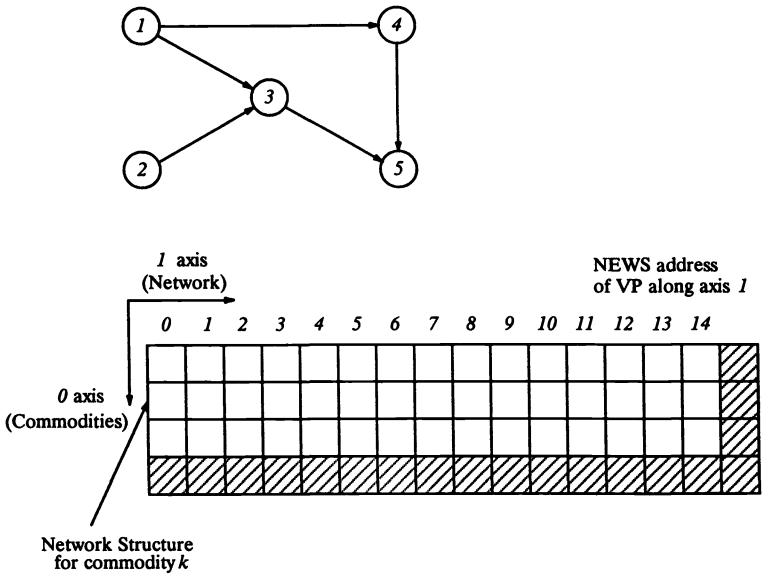


FIG. 1. *Memory configuration of the* FE *and the* CM *for the dense implementation.*

topology for all the commodities, the mapping of arcs into VPs and the partitioning of VPs into segments will be identical for each row of the grid.

The control of the algorithm is identical for each row of the grid (i.e., for each network problem). Row k of the 0-axis will store the data of the network problem for the kth commodity. This configuration is illustrated in Fig. 2. The algorithm iterates along the 1-axis until some convergence criteria is satisfied for all the rows. Once the single commodity networks are solved by iterations along the 1-axis, the algorithm executes Step 2 using scan operations along the 0-axis. (Since the flows of each commodity satisfy $x_k(i,j) \geq 0$ we only need to compute the projection for the upper bound of the GUB constraints.) This step is implemented by the following code segment in C/Paris:

```
CM-spread-with-f-add-1L(scr1, x, 0, S, E);
CM-f-sub-mult-1L(scr2, U, scr1, KW, S, E);
```



Data Fields in the k-th row corresponding to commodity k.

NEWS address of VP along axis 1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Node	1	1	1	2	2	3	3	3	3	4	4	4	5	5	5
Segment bits	1	0	0	1	0	1	0	0	0	1	0	0	1	0	0
Supply Data	$s_k(1)$	$s_k(1)$	$s_k(1)$	$s_k(2)$	$s_k(2)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(3)$	$s_k(4)$	$s_k(4)$	$s_k(4)$	$s_k(5)$	$s_k(5)$	$s_k(5)$
Demand Data	$d_k(1)$	$d_k(1)$	$d_k(1)$	$dx(2)$	$d_k(2)$	$d_k(3)$	$d_k(3)$	$d_k(3)$	$d_k(3)$	$d_k(4)$	$d_k(4)$	$d_k(4)$	$d_k(5)$	$d_k(5)$	$d_k(5)$
Joint Capacity	∞	$U(1,3)U(1,4)$	∞	$U(2,3)$	∞	$U(1,3)U(2,3)U(3,5)$	∞	$U(1,4)U(4,5)$	∞	$U(3,5)U(4,5)$					
Send address in NEWS coordinates along axis 1	0	6	10	3	7	5	1	4	13	9	2	14	12	8	11

FIG. 2. *Representing sparse multicommodity networks on the* CM.

```
CM-f-min-2-1L(scr2, PSI, S, E);
CM-f-subtract-2-1L(PSI, scr2, S, E);
CM-f-divide-2-1L(scr2, W, S, E);
CM-f-add-2-1L(x, scr2, S, E);
```

4. Experimental design and performance evaluation. The quadratic optimization Algorithm 2.2 was implemented on the Connection Machine CM–2 using C/Paris, as explained in the previous section. As pointed out in §2, the steps of Algorithm 2.2 can be executed in any almost-cyclic fashion. Our implementation carries out iteratively Steps 1.1–1.3 until some termination criterion is satisfied for the equality constraints (MINOR iterations). Once this tolerance is achieved, it executes Step 2 (MAJOR iteration) and resumes minor iterations. The algorithm terminates

when both of the following criteria are satisfied:

1. Relative error on GUB constraints for major iterations:

$$100 \times \max_{(i,j)\in\mathcal{E}} \left\{ 0, \frac{\sum_{k=1}^{K} x_k(i,j) - U(i,j)}{U(i,j)} \right\} \leq \epsilon_1. \tag{60}$$

In all experiments we set $\epsilon_1 = 0.1$ percent.

2. Absolute error on network equality constraints for minor iterations:

$$\max_{i\in V_O, j\in V_D, k\in<K>} \left\{ \left| s_k(i) - \sum_{j\in\delta^+(i)} x_k(i,j) \right| , \left| d_k(j) - \sum_{i\in\delta^-(j)} x_k(i,j) \right| \right\} \leq \epsilon_2. \tag{61}$$

In all experiments we set $\epsilon_2 = 10^{-4}$.

In this section we provide a summary of computational results in order to highlight certain aspects of the performance of the algorithm and illustrate its suitability for the solution of very large problems. The program was compiled on a SUN 4/280 FE using compiler flags -O -cm2. We used in all runs a CM–2 with 32-bit floating point accelerators at Thinking Machines Corporation, Cambridge, MA. All times are in seconds. Data input/output is excluded, but time for the transfer of data between the FE and the CM is included, together with all time spent in communications on the CM. All times reported are in total CPU time as recorded by the FE. This includes CM time for execution of the C/Paris code and CPU time for the controlling program and FE calculations. Most runs were carried out on a lightly used FE and the CM times consume more than 95 percent (up to 99 percent in some cases) of the total time.

The dense implementation runs at approximately 1.6 GFLOPS when implemented in C/Paris on a 64K CM–2. It is possible to design an optimal implementation based on the NEWS grid, that minimizes the amount of communication required. Such an implementation was carried out in microcode and is described in McKenna and Zenios [22]; it executes at three GFLOPS rate in solving single commodity problems. Unless otherwise stated, all subsequent experiments are using the C/Paris implementation.

4.1. Problem generator. We wrote a problem generator that provides control over several characteristics of the test problems. The generator was also written in C/Paris on the CM. It is thus possible to generate extremely large problems in memory and pass them on to the solver without the need to transfer data to an external storage device—a task that would take several hours for the bigger problems. The input parameters for the generator are: (1) Number of origin and destination nodes m_O and m_D, respectively; (2) number of commodities K; (3) condition number ρ of the Hessian matrix; (4) largest coefficient, max-c, for linear term; (5) percentage, α, of joint capacity constraints that are active at the optimal solution; (6) the tightness, β, of the active joint capacity constraints; (7) maximum supply or demand, max-sd, at each node. The generator accepts as input the seven control parameters and generates a problem in three steps:

Step 1: Generate K single commodity problems: Set up a two-dimensional NEWS grid of $\lceil m_O \rceil_2 \times \lceil m_D \rceil_2$ active VPs. Generate the objective function coefficients $w_k(i,j)$ in the range [1,ρ] using a uniform random distribution. Generate the objective function coefficients $c_k(i,j)$ in the range [1, max-c]. Generate

supply and demand values, for origin and destination nodes, respectively, in the range [1, max-sd]. Scale all supply values by

$$\frac{\sum_{j=1}^{m_D} d_k(j)}{\sum_{i=1}^{m_O} s_k(i)}$$

to ensure that the equality constraints are feasible.

Step 2: Solve the K uncapacitated, single commodity problems generated in Step 1. (We use the algorithm of Zenios and Censor [31].)

Step 3: Calculate the sum of the optimal flows from Step 2 over all commodities for each arc, say, $ex(i,j)$. Choose, at random, α percent of the arcs that will have active GUB constraints. For the arcs so chosen, set $U(i,j) = \beta \cdot ex(i,j)$. The rest of the arcs are virtually unrestricted, and the joint capacity constraints are set to $U(i,j) = \frac{2}{\beta} \cdot ex(i,j)$.

At this point any arcs that have $ex(i,j) = 0$ are removed from the problem. In general, we found that leaving these arcs in the problem with a large upper bound would make the test problems significantly easier. Some of the problems solved at the early stages of our experimentation kept all the arcs in the problem. Whenever, in subsequent sections, the number of arcs is exactly $m_O \times m_D$, then the problem is relatively easy.

Subsequent sections will describe in detail the parameters used to generate each problem. When no such information is given, then the following base-case test problem is being used: Nodes $m_O = m_D = 256$, commodities K=5, condition number $\rho = 10^2$, largest coefficient for linear term max-c=100, percentage of active joint capacity constraints $\alpha = 10$ percent, tightness of active joint capacity constraints $\beta = 0.90$, maximum supply or demand max-sd=100. When information is provided only for some of the input parameters, then the remaining parameters have the values given here for the base-case.

4.2. Algorithmic performance. The algorithm is a first-order method, and as such it is expected to have a tailing effect with larger steps at the first iterates and smaller steps as it approaches the solution. A small test problem is used to illustrate the performance of the algorithm. Figure 3 plots the absolute maximum error of the GUB constraints $\left(\max_{(i,j)\in\mathcal{E}} \left\{0, \sum_{k=1}^{K} x_k(i,j) - U(i,j)\right\}\right)$ at successive major iterations. Within each major iteration the figure illustrates the absolute error of the equality constraints at selected minor iterations. Following a major iteration, the joint capacity constraint error is zero (recall that Step 2 of the algorithm will satisfy exactly the joint capacity constraints), but the error of the equality constraints is large. Successive minor iterations reduce the error at the equations and increasingly violate the joint capacity constraints. Depending on which constraints are "soft" in a given model (i.e., the GUB constraints or the equality constraints), the results of Fig. 3 provide some guidance on when the algorithm should be terminated.

The question is raised whether the minor iterations should be terminated with a loose tolerance at the early major iterations. For the dense implementation on the CM, such a strategy is not efficient for the following reason: In order to execute the minor iterations, each commodity has to be loaded into the CM memory from the FE. A significant amount of time is consumed in this step. As illustrated in Fig. 4, after the first few major iterations, more time is spent in transferring data from the FE to the CM than in computations. Terminating the algorithm with looser minor iteration tolerance will increase the number of major iterations, and

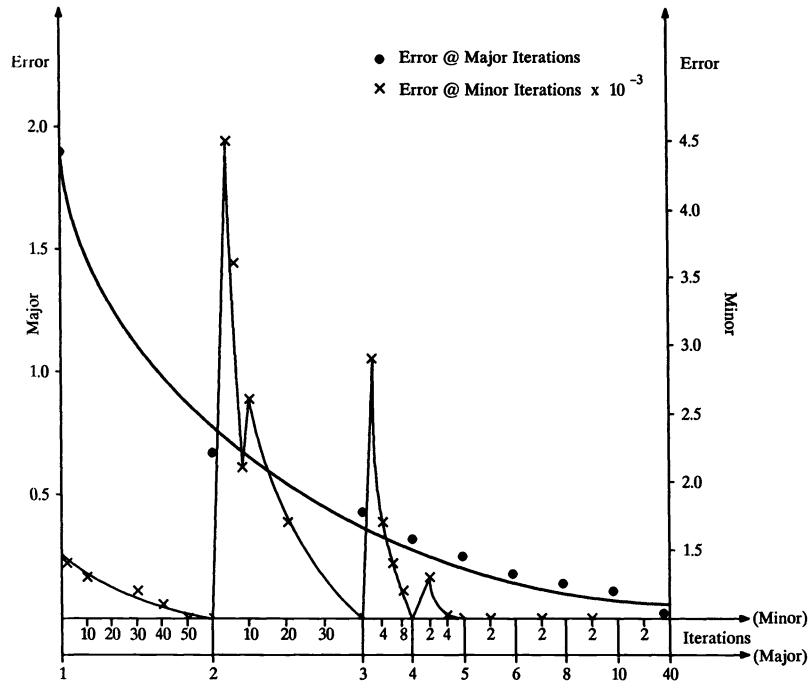


FIG. 3. *Maximum error at successive major and minor iterations.*

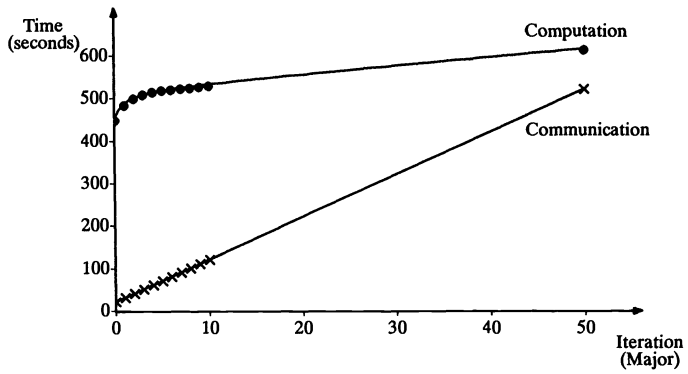


FIG. 4. *Computation and communication times for the dense implementation.*

hence the communication time. After the first 1–2 major iterations, the decrease in computing time will not compensate for the increase of communication time. This is one example where a strategy that would be efficient on a serial computer, or a coarse-grain parallel computer, is not advisable on a massively parallel system. The time spent in communications is critical in the choice of internal algorithmic tactics.

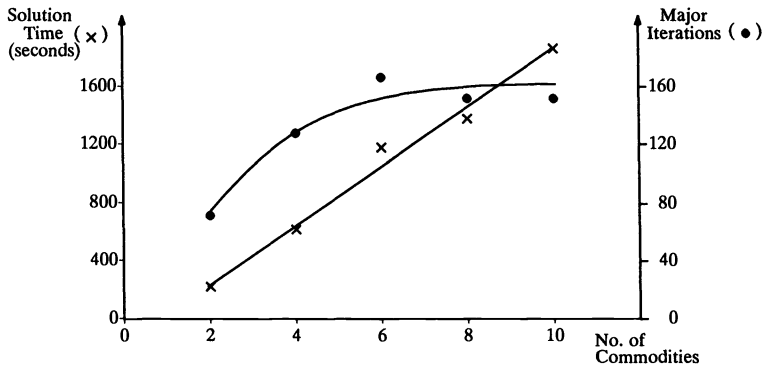


FIG. 5. *Effect of increasing the number of commodities K on the performance of the algorithm.*

4.3. The effects of problem structure. It is well established in the optimization folklore that the performance of a nonlinear programming algorithm may vary substantially for different problem characteristics. In this section we look at the performance of the quadratic programming algorithm when the following problem parameters change: (1) number of commodities K, (2) condition number ρ, (3) percentage α of active GUB constraints, and (4) tightness β of active GUB constraints.

4.3.1. Increasing number of commodities. Figure 5 illustrates both solution time and number of major iterations as the number of commodities increases for a given problem size. Problems are getting only slightly more difficult with increasing number of commodities, as manifested by the small increase in the number of major iterations. However, solution times increase linearly with the number of commodities. This observation provides empirical verification that the amount of computation and communication required per iteration is a linear function of the number of commodities.

There is an interesting implication from the results of this figure: If a massively parallel architecture scales linearly in size with the number of commodities, then larger problems can be solved with only slight increase of solution time. This increase will be proportional to the increase in major iterations.

4.3.2. Increasing condition number. First-order algorithms, like those presented here, are very sensitive to ill-conditioning. Figure 6 illustrates the performance of the algorithm as the condition number of the problem increases, and for problems with varying number of commodities. The adverse impact of ill-conditioning is more significant for problems with more commodities. For condition number $\rho = 10^2$, the algorithm can solve problems with a large number of commodities. (In §4.5, we report solution profiles for problems with up to 20 commodities.) For condition number $\rho = 10^3$, the algorithm can still solve problems with many commodities, but at significant increase in computing time. For condition number $\rho = 10^4$, the algorithm converges very slowly when applied to problems with more than two commodities. The information in this section provides general guidelines on the potential difficulty of a given test problem.

4.3.3. Increasing percentage, α, of active GUB constraints. As the number of joint capacity constraints that are active at the optimal solution increases, the

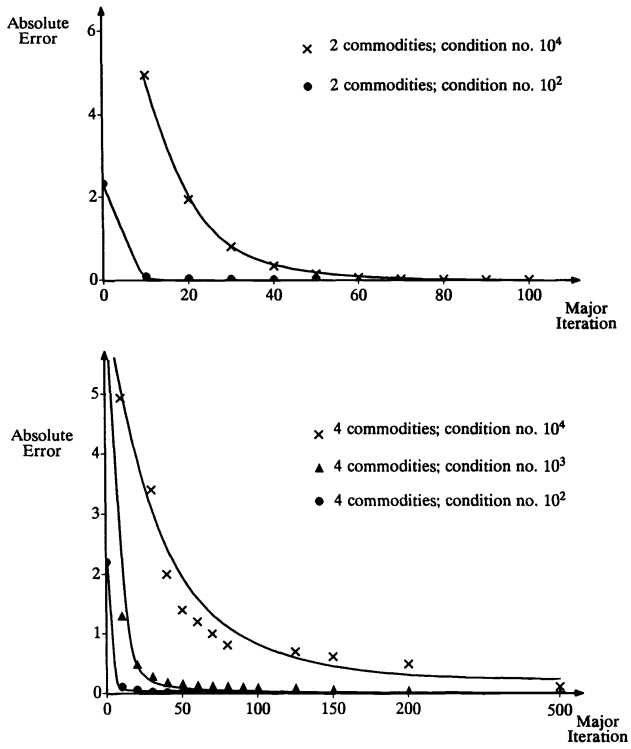


FIG. 6. *Effect of increasing the condition number ρ on the performance of the algorithm.*

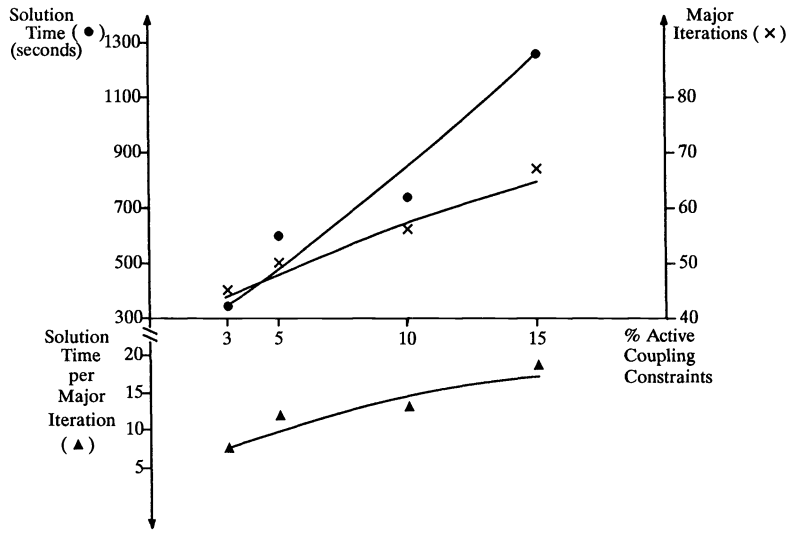


FIG. 7. *Effect of increasing the percentage of active* GUB *constraints α on the performance of the algorithm.*

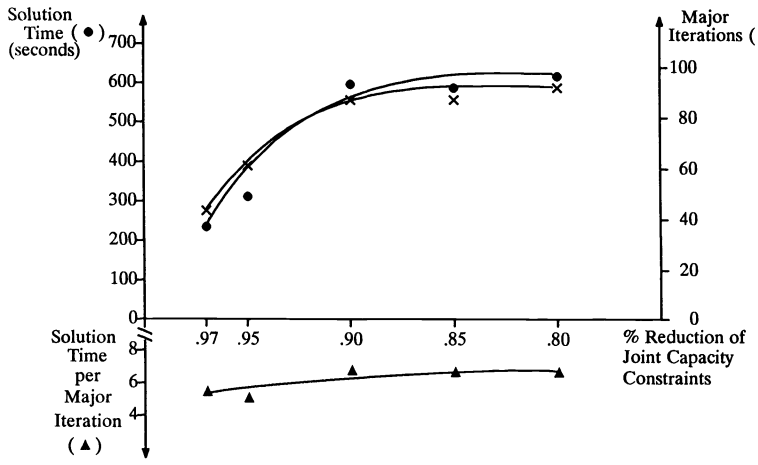


FIG. 8. *Effect of increasing the tightness of the joint capacity constraints β on the performance of the algorithm.*

problems become more difficult. Figure 7 shows the increase in: (1) major iterations, (2) total solution time, and (3) solution time per major iteration, as the percentage α of active constraints increases.

We observe that the increase in solution time is primarily due to the increase in number of major iterations. Hence, no improvements in performance can be anticipated with increases in the size of the computer. It is also usually unknown a priori how many constraints are active at the solution. Hence, the results of this section are of limited practical value. They mainly illustrate two facts. First, the algorithm is capable of solving problems with a significant number of active GUB constraints, and second, the solution time increases only linearly with increase in the number of active constraints.

4.3.4. Increasing tightness, β, of active GUB constraints. Intuitively, one expects the following: If the joint capacity constraint is larger than the sum of the unconstrained optimal flows on all commodities, then the problem is trivial. It will be solved in one major iteration. If the joint capacity is slightly less than the sum of unconstrained flows, then some flow can be diverted easily to adjacent arcs. If the joint capacity is significantly less than the sum of unconstrained flows, then more flow has to be diverted to adjacent arcs and the problem gets more difficult.

This intuition is confirmed in Fig. 8 when the tightness of the active capacity constraints ranges from $\beta = 0.97$ to $\beta = 0.90$. (Recall that for $\beta = 1.00$ the joint capacity constraints are not active, and they become more tight as β decreases.) From the same figure, however, it appears that the problems get marginally more difficult as β decreases to 0.80. Additional experimentation is needed before one attempts to devise an explanation.

4.4. The performance of the sparse implementation. Table 1 summarizes the results in solving identical problems using both the dense and sparse implementations. All test problems solved here are totally dense, and hence this comparison is biased against the sparse implementation. Nevertheless, some interesting observations

TABLE 1
Comparing the dense and sparse implementations. (Solution times in seconds.)

No.	Problem size	Solution characteristics	4K CM–2		8K CM–2
			Dense implementation	Sparse implementation	Sparse implementation
1	8 com. 128 nodes 4096 arcs	VP ratio CM time(sec)	1 144	32 305	16 169
2	8 com. 64 nodes 1024 arcs	VP ratio CM time(sec)	1 130	8 218	4 139
3	8 com. 32 nodes 256 arcs	VP ratio CM time(sec)	1 156	1 25	NA

can be made which are very encouraging for the performance of the sparse implementation. If the computer configuration does not have a sufficient number of processors to store all the commodities simultaneously in their sparse representation, then the dense implementation is superior. The time spent in transferring data from the FE to the CM is compensated by the improved computing performance achieved when the algorithm executes at a low VP ratio. Compare the solution time of Problems 1 and 2 on the 4K CM–2, where the dense implementation is faster. However, the sparse implementation can run faster on a bigger machine, while the dense implementation is already executed at a VP ratio of 1. Compare the solution times for Problems 1 and 2 with the dense implementation running at a VP ratio equal to 1 on a 4K CM–2, and the sparse implementation running on an 8K CM–2 with VP ratios 16 and 4, respectively. The sparse implementation is at par with the dense implementation and could improve even further with an increase in the number of processing elements. When both the dense and sparse implementations run with VP ratio 1, then the sparse implementation can be significantly faster (see the results for Problem 3). Identifying the precise conditions under which the sparse implementation should be preferred is a complex problem. Some preliminary discussion on this topic is given in the working paper version of this article.

4.5. Solving large scale problems. As a final exercise, we generated and solved some very large problems. The results are reported in Table 2. The algorithm achieves a good level of accuracy in number of major iterations that range from 20–100. The largest problems (5–10) have more than one million variables and 10 thousand equations and they are solved well within one hour of wall clock time. The same problem could be solved in less than 10 minutes on a 32K CM–2 and less than five minutes on a fully configured 64K CM–2.

5. Concluding remarks. This paper has developed an algorithm for nonlinear multicommodity transportation problems. The algorithm induces a fine-grain decomposition of the problem: First, by node for each commodity, and then by arc. As a result, it is possible to implement this algorithm on massively parallel computer architectures. Of course, implementations on coarse-grain parallel machines are also possible, although such implementations would not fully exploit the potential of the algorithm.

The algorithm appears effective in solving problems of medium difficulty (con-

TABLE 2
Solving large scale problems. (Time in min:sec.)

No.	Problem size		Cond. No.	Major Itns.	Error		Time	
	Network formulation	Lin. prog. formulation			% GUB const.	Absolute node const.	4K CM–2	32K CM–2 (*)
1	2 com. 256 nodes 12976 arcs	512 eqns. 1672 GUB 25952 vars.	10^4	90	.09	10^{-5}	20:00	3:10
2	2 com. 256 nodes 15276 arcs	512 eqns. 1675 GUB 30534 vars.	10	100	.11	10^{-7}	1:00	0:10
3	5 com. 512 nodes 16370 arcs	1280 eqns. 1634 GUB 83850 vars.	10^2	86	.09	10^{-6}	1:50	:20
4	5 com. 512 nodes 65478 arcs	2560 eqns. 6552 GUB 327435 vars.	10^2	NA	.09	10^{-6}	4:30	0:45
5	5 com. 1024 nodes 262000 arcs	5120 eqns. 26181 GUB 1310000 vars.	10^2	41	.09	10^{-6}	14:10	2:15
6	10 com. 1024 nodes 262144 arcs	10240 eqns. 25957 GUB 2621140 vars.	10^2	37	.09	10^{-6}	31:30	4:55
7	20 com. 1024 nodes 262144 arcs	20480 eqns. 26260 GUB 5424880 vars.	10^2	90	.09	10^{-5}	45:00	7:20
8	2 com. 2048 nodes 1002338 arcs	4096 eqns. 104443 GUB 2004678 arcs.	10^2	17	.08	10^{-6}	14:35	2:15
9	5 com. 2048 nodes 1048074 arcs	10240 eqns. 104088 GUB 5240370 vars.	10^2	19	.07	10^{-6}	40:40	6:20
10	8 com. 2048 nodes 1048570 arcs	16384 eqns. 104521 GUB 8384560 vars.	10^2	22	.09	10^{-6}	29:50	4:40

(*)Note: Time on the 32K CM–2 is estimated, dividing the time on the 4K CM–2 by 6.4. This is the empirically observed improvement in performance as the VP ratio is reduced by a factor of 8, when moving from the 4K to the 32K CM–2.

dition numbers 10–1000) to a good level of accuracy. Nevertheless, as a first-order method it could be severely affected by ill-conditioned problems. Also, the attainment of very high accuracy could come with significant increase in the number of iterations and computer times.

Massively parallel implementations on the Connection Machine CM–2 have been possible even for sparse problems. The implementations are very efficient, and match the parallel nature of the algorithm with the computer architecture of the CM–2. We expect significant improvements in the performance of massively parallel architectures over the coming years. For example, the current version of the CM–2 has processing elements that operate at 7MHZ; this is significantly lower than the operating cycle of personal computers. Also, 64-bit WEITEK floating-point accelerators are now being added to some configurations. The algorithm developed here could then achieve higher accuracy with little additional effort. The current model of C/Paris—based on fieldwise representation of data—is far from being the most efficient one for the CM–2. The optimization algorithms developed here should be able to benefit from

improvements in the computer models without additional effort on our part.

It is a severe limitation that these algorithms are applicable only to nonlinear problems. Nevertheless, they are the building blocks for solving linear programs: The quadratic optimization algorithms in the context of proximal minimization of Rockafellar [26] and the entropy optimization extensions in the context of the PMD framework developed by Censor and Zenios [13]. This is the topic of a current study.

As a postscript, we add that a massively parallel algorithm has been developed by Nielsen and Zenios [23] for stochastic programming problems with network recourse. Experiments with that algorithm on the Connection Machine CM–2 for some financial modeling applications reinforce the encouraging results reported here for multicommodity flows.

Acknowledgments. I thank Professor Yair Censor for numerous illuminating discussions on row-action algorithms and for his constant encouragement. I also benefited from the comments of Jill Mesirov and Soren Nielsen. Computing resources for this paper were made available by the North-east Parallel Architectures Center (NPAC) of Syracuse University, Syracuse, NY 13244, and the Army High Performance Computing Research Center (AHPCRC) at the University of Minnesota.

REFERENCES

[1] A. A. ASSAD, *Multicommodity network flows—a survey*, Networks, 8 (1978) pp. 37–91.

[2] D. P. BERTSEKAS, *Algorithms for nonlinear multicommodity network flow problems*, in International Symposium on Systems Optimization and Analysis, A. Bensoussan and J.L. Lions, eds., Springer-Verlag, Berlin, New York, 1979, pp. 210–224.

[3] D. P. BERTSEKAS AND J. N. TSITSIKLIS, *Parallel and Distributed Computation: Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ, 1989.

[4] A. BJÖRCK AND T. ELFVING, *Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[5] G. E. BLELLOCH, *Vector Models for Data-Parallel Computing*, The MIT Press, Cambridge, MA, 1990.

[6] R. BRAMLEY AND A. SAMEH, *Row projection methods for large nonsymmetric linear systems*, CSRD Report 957, Department of Computer Science, University of Illinois, Urbana, IL, January 1990.

[7] L. M. BREGMAN, *The relaxation method for finding the common point of convex sets and its application to the solution of problems in convex programming*, USSR Comput. Math. and Math. Phys., 7 (1967), pp. 200–217.

[8] Y. CENSOR, *Parallel application of block-iterative methods in medical imaging and radiation therapy*, Math. Programming, 42 (1988), pp. 307–325.

[9] ———, *Row-action methods for huge and sparse systems and their applications*, SIAM Rev., 23 (1981), pp. 444–464.

[10] Y. CENSOR AND A. LENT, *An iterative row-action method for interval convex programming*, J. Optim. Theory Appl., 34 (1981), pp. 321–353.

[11] Y. CENSOR, A. R. DE PIERRO, T. ELFVING, G. T. HERMAN, AND A. N. IUSEM, *On iterative methods for linearly constrained entropy maximization*, in Numerical Analysis and Mathematical Modelling, Vol. 24, A. Wakulicz, ed., Banach Center Publications, PWN—Polish Scientific Publisher, Warsaw, Poland, 1990, pp. 145–163.

[12] Y. CENSOR AND S. A. ZENIOS, *Interval constrained matrix balancing*, Linear Algebra Appl., 150 (1991), pp. 393–421.

[13] ———, *The proximal minimization algorithm with D-functions*, J. Optim. Theory Appl., 1991, to appear.

[14] R. J. CHEN AND R. R. MEYER, *Parallel optimization for traffic assignment*, Math. Programming, 42 (1988), pp. 327–345.

[15] J. ECKSTEIN, *Implementing and running the alternating step method on the Connection Machine* CM–2, ORSA J. Comput., 1992, to appear.

[16] T. ELFVING, *An algorithm for maximum entropy image reconstruction from noisy data*, Math. Comput. Modelling, 12 (1989), pp. 729–745.

[17] R. GALLAGER, *A minimum delay routing algorithm using distributed computation*, IEEE Trans. Comm., 25 (1977), pp. 73–85.
[18] M. GONDRAN AND M. MINOUX, *Graphs and Algorithms*, John Wiley and Sons, New York, 1986.
[19] G. T. HERMAN, *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.
[20] W. D. HILLIS, *The Connection Machine*, The MIT Press, Cambridge, MA, 1985.
[21] J. L. KENNINGTON, *A survey of linear cost multicommodity network flows*, Oper. Res., 26 (1978), pp. 209–236.
[22] M. MCKENNA AND S. A. ZENIOS, *An optimal parallel implementation of a quadratic transportation algorithm*, in Fourth SIAM Conference on Parallel Processing for Scientific Computing, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1990, pp. 357–363.
[23] S. NIELSEN AND S. A. ZENIOS, *Data Structures for Network Algorithms on Massively Parallel Architectures*, Report 90–12–07, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990; Parallel Comput., to appear.
[24] ———, *Massively Parallel Algorithms for Nonlinear Stochastic Network Problems*, Report 90–09–08, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, PA, 1990; Oper. Res., to appear.
[25] A. R. DE PIERRO AND A. N. IUSEM, *A relaxed version of Bregman's method for convex programming*, J. Optim. Theory Appl., 5 (1986), pp 421–440.
[26] R. T. ROCKAFELLAR, *Augmented Lagrangians and applications to proximal point algorithms in convex programming*, Math. Oper. Res., 1 (1976), pp. 97–116.
[27] G. L. SCHULTZ AND R. R. MEYER, *A structured interior point method*, SIAM J. Optimization, this issue, pp. 583–602.
[28] S. NIELSEN AND S. A. ZENIOS, *Massively parallel algorithms for singly constrained nonlinear programs*, ORSA J. Comput., 4 (1992), to appear.
[29] R. D. WOLLMER, *Multicommodity networks with resource constraints: the generalized multicommodity flow problem*, Networks, 1 (1972), pp. 245–263.
[30] S. A. ZENIOS AND R. A. LASKEN, *Nonlinear network optimization on a massively parallel Connection Machine*, Ann. Oper. Res., 14 (1988), pp. 147–165.
[31] S. A. ZENIOS AND Y. CENSOR, *Massively parallel row-action algorithms for some nonlinear transportation problems*, SIAM J. Optimization, 1 (1991), pp. 373–400.
[32] ———, *Parallel computing with block-iterative image reconstruction algorithms*, Appl. Numer. Math., 7 (1991), pp. 399–415.
[33] M. C. PINAR AND S. A. ZENIOS, *Parallel decomposition of multicommodity network flow problems*, ORSA J. Comput., 4 (1992), to appear.