CENTER FOR
PARALLEL OPTIMIZATION

PARALLEL VARIABLE DISTRIBUTION

by

M. C. Ferris & O. L. Mangasarian

Computer Sciences Technical Report #1175

August 1993

Parallel Variable Distribution[*]

M. C. Ferris[†] & O. L. Mangasarian[†]

August 1993

Abstract

Variables of an optimization problem are distributed among p processors so that each processor has primary responsibility for updating its own block of variables while allowing the remaining variables to change in a restricted fashion (e. g. along a steepest descent, quasi–Newton, or any arbitrary direction). This parallelization step is followed by a fast synchronization step wherein the affine hull of the points computed by the parallel processors and the current point are searched for an optimal point. Convergence to a stationary point under continuous differentiability is established for the unconstrained case, as well as a linear convergence rate under the additional assumption of a Lipschitzian gradient and strong convexity. For problems constrained to lie in the cartesian product of closed convex sets, convergence is established to a point satisfying a necessary optimality condition under Lipschitz continuous differentiability of the objective function. For problems with more general constraints, convergence is established under stronger conditions. Encouraging computational results on the Thinking Machines CM-5 Multiprocessor on a subset of the publicly available CUTE set of nonlinear programming test problems are given.

[*] This material is based on research supported by the Air Force Office of Scientific Research Grant AFOSR–89–0410 and National Science Foundation Grants CCR–9157632, CCR–9101801 and CDA–9024618

[†] Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, Wisconsin 53706, email: *ferris@cs.wisc.edu, olvi@cs.wisc.edu.*

1 Introduction

We consider the problem

$$\min_{x\in X} f(x) \tag{1.1}$$

where X is a nonempty closed convex set in the n–dimensional real space $\mathbb{R}^n$ and $f\colon \mathbb{R}^n \to \mathbb{R}$ has continuous first partial derivatives on $\mathbb{R}^n$. Our objective in this paper is to distribute p blocks $x_1, \ldots, x_p$ of the variable x, where $x_l \in \mathbb{R}^{n_l}$, $\sum\limits_{l=1}^{p} n_l = n$, among p processors. Given $x^i \in \mathbb{R}^n$, processor l has primary responsibility for updating block $x^i_l \in \mathbb{R}^{n_l}$ of the iterate x^i by solving the following problem.

Parallelization:

$$\begin{array}{rcl} (y^i_l, \lambda^i_{\bar{l}}) & \in & \arg\min_{x_l,\lambda_{\bar{l}}} f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}) \\ & & \text{such that } (x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}) \in X \\ x^{i_l} & := & (y^i_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda^i_{\bar{l}}) \end{array} \tag{1.2}$$

Here $\bar{l}$ denotes the complement of l in $\{1, \ldots, p\}$, $\lambda_{\bar{l}} \in \mathbb{R}^{p-1}$. The matrix $D^i_{\bar{l}} \in \mathbb{R}^{n_{\bar{l}}\times(p-1)}$ is an $n_{\bar{l}}$–by–$(p-1)$ matrix. It is formed by taking an arbitrary direction $d^i \in \mathbb{R}^n$, breaking it into blocks of $d^i_s \in \mathbb{R}^{n_s}$ consistent with the distribution of the variables x_s and placing these vectors along the block diagonal of $D^i_{\bar{l}}$ as follows:

$$D^i_{\bar{l}} := \begin{bmatrix} d^i_1 & & & & & & & \\ & d^i_2 & & & & & & \\ & & \ddots & & & & & \\ & & & d^i_{l-1} & & & & \\ & & & & d^i_{l+1} & & & \\ & & & & & d^i_{l+2} & & \\ & & & & & & \ddots & \\ & & & & & & & d^i_p \end{bmatrix}. \tag{1.3}$$

Once the points x^{i_l}, $l = 1, \ldots, p$, have been computed by the p parallel processors, the affine hull of the points $\{x^i, x^{i_1}, \ldots, x^{i_p}\}$ is typically searched for a best point by a synchronization step as follows.

Synchronization:

$$\begin{array}{rcl} x^{i+1} & = & \mu_0^i x^i + \sum_{k=1}^{p} \mu_k^i x^{i_k} \\ (\mu_0^i, \mu_1^i, \dots \mu_p^i) & \in & \arg\min_{\mu_0,\mu_1,\dots\mu_p} f(\mu_0 x^i + \sum_{k=1}^{p} \mu_k x^{i_k}) \\ & & \text{such that} \quad \begin{array}{l} \mu_0 x^i + \sum_{k=1}^{p} \mu_k x^{i_k} \in X \\ \mu_0 + \sum_{k=1}^{p} \mu_k = 1. \end{array} \end{array} \tag{1.4}$$

We note immediately that a fundamental difference between our method and that of block Jacobi for solving nonlinear equations [4] and coordinate descent [18] is the presence of the "forget–me–not" term $x_{\bar{l}}^i + D_{\bar{l}}^i \lambda_{\bar{l}}$ in problem (1.2). Thus in contrast with block Jacobi and coordinate descent where the coordinates of the blocks of variables $x_{\bar{l}}$ handled by the other processors are held fixed, our parallel subproblems (1.2) allow these blocks of variables to move in arbitrary directions $D_{\bar{l}}^i$, typically generated using steepest descent or quasi–Newton directions in the space of these variables. This novel idea allows each parallel processor to obtain a better minimum and endows the algorithm with a robustness property that makes it difficult to fail.

We outline the contents of the paper now. In Section 2 we consider the unconstrained case and establish convergence of our algorithm by using ideas similar to those of the parallel gradient algorithm proposed in [12]. We also establish a linear convergence rate for the unconstrained algorithm. In Section 3 we consider problems with block separable constraints, that is where the constraints are a cartesian product of closed convex sets. In Section 4 we consider more general constraints and in Section 5 we give some preliminary computational results. We conclude with some brief remarks in Section 6.

We briefly describe our notation. The sequence $\{x^i\}$, $i = 1, 2, \dots$, will represent iterates in the n–dimensional real space $\mathbb{R}^n$ generated by some algorithm. For $l = 1, \dots , p$, $x_l^i \in \mathbb{R}^{n_l}$ will represent an n_l–dimensional subset of the components of x^i, where $\sum_{l=1}^{p} n_l = n$. The complement of l in $\{1, \dots , p\}$ will be denoted by $\bar{l}$ and we write $x^i = (x_l^i, x_{\bar{l}}^i)$, $l = 1, \dots , p$. For a differentiable function $f\colon\ \mathbb{R}^n \to \mathbb{R}$, ∇f will denote the n–dimensional (row) vector of partial derivatives with respect to x, and $\nabla_l f$ will denote the n_l–dimensional vector of partial derivatives with respect to $x_l \in \mathbb{R}^{n_l}$, $l = 1, \dots , p$. If f has continuous first partial derivatives on $\mathbb{R}^n$, we say $f \in C^1(\mathbb{R}^n)$. If f has Lipschitz continuous first partial derivatives on $\mathbb{R}^n$ with constant $K > 0$, that is:

$$\|\nabla f(y) - \nabla f(x)\| \leq K \|y - x\|, \ \ \forall x, y \in \mathbb{R}^n$$

we write $f \in LC_K^1(\mathbb{R}^n)$. Unless stated otherwise, $\|\cdot\|$ denotes the Euclidean norm, that is $\|x\| = \sqrt{x^T x}$, for x in a finite–dimensional space of appropriate dimension. A differentiable function $f\colon\ \mathbb{R}^n \to \mathbb{R}$, is strongly convex on $\mathbb{R}^n$ with constant $k > 0$ if

$$f(y) - f(x) - \nabla f(x)(y - x) \geq \frac{k}{2} \|y - x\|^2, \ \ \forall x, y \in \mathbb{R}^n$$

or equivalently

$$(\nabla f(y) - \nabla f(x))(y-x) \geq k\left\|y-x\right\|^2, \ \ \forall x, y \in \mathbb{R}^n .$$

We adapt the definition of a forcing function [15, p. 479] for our purposes.

Definition 1.1 ***Forcing Function:*** *A **forcing function** σ is a continuous function from the nonnegative real line $\mathbb{R}_+$ into itself such that $\sigma(0) = 0$, $\sigma(\xi) > 0$, for $\xi > 0$ and for every sequence of nonnegative real numbers $\{\xi_i\}$:*

$$\sigma(\xi_i) \longrightarrow 0 \textit{ implies } \xi_i \longrightarrow 0.$$

Some typical examples of forcing functions are

$$\gamma\xi, \gamma\xi^2, \max\{\sigma_1(\xi), \sigma_2(\xi)\}, \min\{\sigma_1(\xi), \sigma_2(\xi)\} \text{ and } \sigma_2(\sigma_1(\xi)),$$

where γ is a positive number and $\sigma_1(\xi)$ and $\sigma_2(\xi)$ are forcing functions.

2 Unconstrained PVD

We begin by stating and establishing convergence of a parallel distribution algorithm for the unconstrained minimization problem (1.1), where $X = \mathbb{R}^n$. In this case, we define a stationary point x as any point such that $\nabla f(x) = 0$.

Theorem 2.1 ***Unconstrained PVD Algorithm:*** *Let $f \in LC^1_K(\mathbb{R}^n)$. Start with any $x^0 \in \mathbb{R}^n$. Having x^i, stop if $\nabla f(x^i) = 0$. Otherwise, compute x^{i+1} as follows:*

Parallelization: *For $l = 1, \ldots, p$ compute $x^{i_l} \in \mathbb{R}^n$ such that*

$$\begin{array}{rcl} (y^i_l, \lambda^i_{\bar{l}}) & \in & \arg\min_{x_l, \lambda_{\bar{l}}} f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}) \\ x^{i_l} & := & (y^i_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda^i_{\bar{l}}) \end{array} \tag{2.1}$$

Here $D^i_{\bar{l}}$ is an $n_{\bar{l}}$–by–$(p-1)$ matrix generated from an arbitrary direction $d^i \in \mathbb{R}^n$ as in (1.3).
Typically, $d^i = -\nabla f(x^i) / \left\|\nabla f(x^i)\right\|$.

Synchronization: *Compute $x^{i+1} \in \mathbb{R}^n$ such that*

$$f(x^{i+1}) \leq \min_{l=1,\cdots,p} f(x^{i_l}). \tag{2.2}$$

Typically, x^{i+1} is computed as in (1.4).

Convergence: *For a bounded sequence $\{d^i\}$, either the sequence $\{x^i\}$ terminates at a stationary point $x^{\bar{i}}$ or each of its accumulation points is stationary and* $\lim_{i\to\infty} \nabla f(x^i) = 0$.

Proof For $l = 1, \ldots, p$ define the auxiliary functions $\theta^i_l \colon \mathbb{R}^{n_l+(p-1)} \to \mathbb{R}$, as follows:

$$\theta^i_l(x_l, \lambda_{\bar{l}}) := f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}). \tag{2.3}$$

Then

$$\nabla\theta^i_l(x_l, \lambda_{\bar{l}}) = \left[\nabla_l f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}), \nabla_{\bar{l}} f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}})D^i_{\bar{l}}\right]. \tag{2.4}$$

Since f has a Lipschitz continuous gradient and $\{d^i\}$ is bounded, it follows that θ^i_l also has a Lipschitz continuous gradient with constant K_1, say. Define now, for $l = 1, \ldots, p$, $(z^i_l, \nu^i_l) \in \mathbb{R}^{n_l+(p-1)}$ as follows

$$(z^i_l, \nu^i_l) := (x^i_l, 0) - \frac{1}{K_1}\nabla\theta^i_l(x^i_l, 0). \tag{2.5}$$

By the Quadratic Bound Lemma A.2, it follows that for $l = 1, \ldots, p$

$$\theta^i_l(x^i_l, 0) - \theta^i_l(z^i_l, \nu^i_l) \geq \frac{1}{2K_1}\left\|\nabla\theta^i_l(x^i_l, 0)\right\|^2. \tag{2.6}$$

But by the PVD algorithm parallelization step (2.1)

$$\theta^i_l(y^i_l, \lambda^i_{\bar{l}}) \leq \theta^i_l(z^i_l, \nu^i_l). \tag{2.7}$$

Hence the last two inequalities yield for $l = 1, \ldots, p$

$$\theta^i_l(x^i_l, 0) - \theta^i_l(y^i_l, \lambda^i_{\bar{l}}) \geq \frac{1}{2K_1}\left\|\nabla\theta^i_l(x^i_l, 0)\right\|^2, \tag{2.8}$$

or

$$f(x^i) - f(y^i_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda^i_{\bar{l}}) \geq \frac{1}{2K_1}\left\|\nabla\theta^i_l(x^i_l, 0)\right\|^2 \geq \frac{1}{2K_1}\left\|\nabla_l f(x^i)\right\|^2. \tag{2.9}$$

By noting the definition of x^{i_l} in (2.1) we have

$$f(x^i) - f(x^{i_l}) \geq \frac{1}{2K_1}\left\|\nabla_l f(x^i)\right\|^2, \quad l = 1, \ldots, p. \tag{2.10}$$

Hence summing over l and dividing by p gives

$$f(x^i) - \frac{1}{p}\sum_{l=1}^{p} f(x^{i_l}) \geq \frac{1}{2pK_1}\sum_{l=1}^{p}\left\|\nabla_l f(x^i)\right\|^2 = \frac{1}{2pK_1}\left\|\nabla f(x^i)\right\|^2. \tag{2.11}$$

But from the synchronization step (2.2) we have the inequality

$$f(x^{i+1}) \leq \frac{1}{p}\sum_{l=1}^{p} f(x^{i_l}). \tag{2.12}$$

Combining the last two inequalities gives

$$f(x^i) - f(x^{i+1}) \geq \frac{1}{2pK_1}\left\|\nabla f(x^i)\right\|^2. \tag{2.13}$$

By the PVD algorithm $\{x^i\}$ terminates at an $x^{\bar{i}}$ only if $\nabla f(x^{\bar{i}}) = 0$. Suppose now $\nabla f(x^{\bar{i}}) \neq 0$ for all i and that $\{x^{i_j}\}$ converges to $\bar{x}$. Since the sequence $\{f(x^i)\}$ is nonincreasing by (2.13) and has an accumulation point $f(\bar{x})$, it converges to $f(\bar{x})$, by Lemma A.1. By (2.13) we have that

$$0 = \lim_{j\to\infty} (f(x^{i_j}) - f(x^{i_j+1})) \geq \lim_{j\to\infty} \frac{1}{2pK_1} \left\|\nabla f(x^{i_j})\right\|^2. \tag{2.14}$$

Hence $\lim_{j\to\infty} \|\nabla f(x^{i_j})\|^2$ and $\nabla f(\bar{x}) = 0$. The limiting property of the sequence $\{\nabla f(x^i)\}$ also follows from (2.13). □

If we further assume that the function f is strongly convex on $\mathbb{R}^n$ then the PVD sequence $\{x^i\}$ converges linearly. We state the result as the following theorem.

Theorem 2.2 ***Linear convergence of PVD algorithm:*** *If in addition to the assumptions of Theorem 2.1 the function f is strongly convex with constant $k > 0$, then the sequence of iterates $\{x^i\}$ generated by the PVD algorithm converges linearly to the unique solution $\bar{x}$ of (1.1) at the linear root rate*

$$\left\|x^i - \bar{x}\right\| \leq \left(\frac{2}{k}(f(x^0) - f(\bar{x}))\right)^{\frac{1}{2}} \left(1 - \frac{1}{p}\left(\frac{k}{K_1}\right)^2\right)^{\frac{i}{2}} \tag{2.15}$$

Proof The proof follows from (2.13) and the linear convergence theorem, Theorem A.3 of Appendix A. □

We turn our attention now to optimization problems with block separable constraint sets.

3 PVD with Block Separable Constraints

We consider the following problem in this section:

$$\min_{x\in X} f(x) = \min_{\substack{x_l \in X_l \\ l=1,\dots,p}} f(x_1, \dots, x_p) \tag{3.1}$$

That is, $x \in \prod_{l=1}^{p} X_l$, where X_l, $l = 1, \dots, p$ are closed convex sets in $\mathbb{R}^{n_l}$, $l = 1, \dots, p$, $\sum_{l=1}^{p} n_l = n$.

Before we specify our PVD algorithm for this above problem, we need a few definitions and some preliminary results. We begin with the concept of an optimality function.

Definition 3.1 ***Optimality function:*** *For the problem $\min_{x\in X} f(x)$, $f \colon \mathbb{R}^n \to \mathbb{R}$ and $X \subseteq \mathbb{R}^n$, the function $\phi \colon X \to \mathbb{R}$ is an* ***optimality function*** *if :*

(i) $\phi(x) \geq 0$ on X.

(ii) $x \in \arg\min_{x\in X} f(x) \implies \phi(x) = 0.$

(iii) ϕ *is nonconstant and lower semicontinuous on* X*.*

A ***stationary point*** *for the optimality function is an* $x \in X$ *such that* $\phi(x) = 0$*.*

The nonconstancy condition is merely to rule out the trivial case of $\phi(x) \equiv 0$. If $X = \mathbb{R}^n$ and $f \in C^1(\mathbb{R}^n)$, then $\phi(x) := \|\nabla f(x)\|$ is an optimality function for $\min_{x\in\mathbb{R}^n} f(x)$, and the notion of a stationary point is standard. If X is a closed convex set in $\mathbb{R}^n$, then the following function determined from the minimum principle [11, 7] serves as the *minimum principle* optimality function:

$$\phi(x) := -\min_{d} \left\{\nabla f(x)d \mid x + d \in X, \|d\|_\infty \leq \alpha\right\}, \quad \text{for some } \alpha > 0. \tag{3.2}$$

A simple argument shows that ϕ is lower semicontinuous on X when $f \in C^1(\mathbb{R}^n)$. Furthermore, if $\bar{x}$ is a stationary point for ϕ, then

$$0 = \min_{y\in X} \left\{\nabla f(\bar{x})(y - \bar{x}) \mid \|y - \bar{x}\|_\infty \leq \alpha\right\}.$$

Hence, $\bar{x} \in \arg\min_{y\in X} \{\nabla f(\bar{x})(y - \bar{x}) \mid \|y - \bar{x}\|_\infty \leq \alpha\}$. Since the objective function is linear in y and X is convex, it follows that $\bar{x} \in \arg\min_{y\in X} \nabla f(\bar{x})(y - \bar{x})$ also, or equivalently that

$$\bar{x} \in X \text{ and } \nabla f(\bar{x})(y - \bar{x}) \geq 0, \;\; \forall y \in X. \tag{3.3}$$

This is precisely the minimum principle necessary optimality condition and we have

Lemma 3.2 *If* $\bar{x}$ *is a stationary point for the optimality function* ϕ *given in (3.2), then* $\bar{x}$ *satisfies the minimum principle (3.3).*

We show now that using an Armijo stepsize rule [1] along a bounded Frank–Wolfe algorithm direction [9] produces a function decrease that dominates the minimum principle optimality function (3.2). This relationship will be needed in establishing the convergence of our PVD algorithm. However, we emphasize that the PVD algorithm does not employ either the Frank–Wolfe algorithm or the Armijo stepsize.

Lemma 3.3 *For the problem* $\min_{x\in X} f(x)$ *where* X *is a convex subset of* $\mathbb{R}^n$ *and* $f \in LC^1_K(\mathbb{R}^n)$ *consider the following direction–stepsize procedure for a given* $x \in X$*:*

Direction (Frank–Wolfe):

$$d \in \arg\min_{\substack{d\in X-x \\ \|d\|_\infty \leq \alpha}} \nabla f(x)d. \tag{3.4}$$

Stepsize (Armijo): $\nu = \max\left\{1, \frac{1}{2}, \frac{1}{4}, \ldots\right\}$ *such that*

$$f(x) - f(x + \nu d) \geq -\frac{\nu}{2}\nabla f(x)d \tag{3.5}$$

Then for

$$x^+ := x + \nu d \tag{3.6}$$

it follows that

$$f(x) - f(x^+) \geq \sigma(\phi(x)) \tag{3.7}$$

where ϕ is the minimum principle optimality function defined in (3.2), σ is the forcing function given by

$$\sigma(\xi) := \min\left\{\frac{1}{2}\xi, \frac{1}{4\alpha^2 K}\xi^2\right\}. \tag{3.8}$$

Proof By the Quadratic Bound Lemma A.2

$$\begin{aligned} f(x) - f(x + \nu d) &\geq \nu(-\nabla f(x)d - \frac{K\nu}{2}\left\|d\right\|^2) \\ &= \nu(-\frac{1}{2}\nabla f(x)d - \frac{K\nu}{2}\left\|d\right\|^2 - \frac{1}{2}\nabla f(x)d) \\ &\geq -\frac{\nu}{2}\nabla f(x)d \end{aligned}$$

whenever $\frac{K\nu}{2}\left\|d\right\|^2 \leq -\frac{1}{2}\nabla f(x)d$. Hence by the stepsize choice, either $\nu = 1$, or else when ν is replaced by 2ν the last inequality is violated, that is

$$K\nu\left\|d\right\|^2 > -\frac{1}{2}\nabla f(x)d.$$

Since the direction generated by (3.4) satisfies $\|d\| \leq \alpha$, it follows from the last inequality and the possibility that $\nu = 1$, that

$$\nu \geq \min\left\{1, -\frac{1}{2\alpha^2 K}\nabla f(x)d\right\}. \tag{3.9}$$

Using this lower bound on ν in (3.5) gives the desired inequality (3.7), after replacing $-\nabla f(x)d$ by $\phi(x)$ using (3.2). □

Before stating and proving convergence of our PVD algorithm we need to define a distributed optimality function over the block separable constraint set.

Definition 3.4 ***Distributed optimality function:*** *For the problem (3.1) we define the* ***distributed optimality function***

$$\phi(x) := \sum_{l=1}^{p}\phi_l(x) \tag{3.10}$$

where $\phi_l \colon X \to \mathbb{R}$ and for $l = 1, \ldots, p$

(i) $\phi_l(x) \geq 0$ on X.

(ii) $x \in \arg\min_{y_l \in X_l} f(y_l, x_{\bar{l}}) \implies \phi_l(x) = 0$.

(iii) ϕ_l *is nonconstant and lower semicontinuous on* X.

A ***stationary point*** *for the optimality function (3.10) is an* $x \in X$ *such that* $\phi(x) = 0$.

We note that

$$x \in \arg\min_{x \in X} f(x) \implies \phi(x) = \sum_{l=1}^{p} \phi_l(x) = 0.$$

An example of a distributed optimality function for problem (3.1) is the following:

$$\phi_l(x) := - \min_{\substack{d_l \in X_l - x_l \\ \|d_l\|_\infty \le \alpha}} \nabla_l f(x) d_l. \tag{3.11}$$

The following lemma shows that a forcing function for the distributed optimality function (3.10) also forces any constituent partial optimality function to zero.

Lemma 3.5 *If* σ *is a forcing function, then for any* $J \subseteq \{1, \ldots, p\}$,

$$\sigma(\phi(x^i)) \to 0 \implies \sum_{l \in J} \phi_l(x^i) \to 0,$$

where ϕ *is the distributed optimality function given by (3.10).*

Proof Since σ is a forcing function it follows that

$$\sigma(\phi(x^i)) \to 0 \implies \phi(x^i) = \sum_{l=1}^{p} \phi_l(x^i) \to 0.$$

However, by definition $\phi_l(x^i) \ge 0$, from which the result follows. □

We are ready to state our PVD algorithm for (3.1) and establish its convergence.

Theorem 3.6 ***Block Constrained PVD Algorithm:*** *Let* $f \in LC^1_K(\mathbb{R}^n)$ *and let* X_l *be nonempty convex sets in* $\mathbb{R}^{n_l}$ *for* $l = 1, \ldots, p$. *Start with any* $x^0 \in \mathbb{R}^n$. *Having* x^i, *stop if* $\phi(x^i) = 0$. *Otherwise, compute* x^{i+1} *as follows:*

Parallelization: *For* $l = 1, \ldots, p$ *compute* $x^{i_l} \in \mathbb{R}^n$ *such that*

$$\begin{array}{rcl} (y^i_l, \lambda^i_{\bar{l}}) & \in & \arg\min_{x_l, \lambda_{\bar{l}}} f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}} \lambda_{\bar{l}}) \\ & & \textit{such that } x_l \in X_l, \;\; x^i_{\bar{l}} + D^i_{\bar{l}} \lambda_{\bar{l}} \in \prod_{\substack{s=1 \\ s \ne l}}^{p} X_s \\ x^{i_l} & := & (y^i_l, x^i_{\bar{l}} + D^i_{\bar{l}} \lambda^i_{\bar{l}}) \end{array} \tag{3.12}$$

Synchronization: *Compute* $x^{i+1} \in X$ *such that*

$$f(x^{i+1}) \le \min_{l=1,\ldots,p} f(x^{i_l}). \tag{3.13}$$

Convergence: *For a bounded sequence $\{d^i\}$, either the sequence $\{x^i\}$ terminates at a stationary point $x^{\bar{i}}$ or every accumulation point of $\{x^i\}$ is stationary, that is it satisfies the minimum principle necessary optimality condition (3.3) and* $\lim_{i\to\infty}\phi(x^i)=0$.

Proof First of all, note that by choosing $x_l = x^i_l$ and $\lambda_{\bar{l}} = 0$ in (3.12), it follows that $f(x^{i_l}) \le f(x^i)$ for $l = 1,\dots,p$. Furthermore, $f(x^{i+1}) \le f(x^{i_l})$ for all l by (3.13). Hence the sequence $\{f(x^i)\}$ is nonincreasing. If $\{x^i\}$ does not terminate at a stationary point $x^{\bar{i}}$ of the minimum principle optimality function, then let $\{x^{i_j}\}$ converge to $\bar{x}$. Since $\{f(x^i)\}$ has an accumulation point $f(\bar{x})$, it must converge to $f(\bar{x})$ by Lemma A.1. In particular, it follows that $f(x^i) - f(x^{i+1}) \to 0$. To complete the proof of the theorem, we will show that this implies that $\phi(x^i) \to 0$. Since ϕ is lower semicontinuous it follows that $\phi(\bar{x}) = 0$ and by Lemma 3.2, the minimum principle (3.3) is satisfied at $\bar{x}$.

As in the proof of Theorem 2.1, we define the auxiliary functions $\theta^i_l\colon \mathbb{R}^{n_l+(k-1)} \to \mathbb{R}$ for $l = 1,\dots,p$ by

$$\theta^i_l(x_l,\lambda_{\bar{l}}) := f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}),$$

with gradient determined by

$$\nabla\theta^i_l(x_l,\lambda_{\bar{l}}) = \left[\nabla_l f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}), \nabla_{\bar{l}} f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}})D^i_{\bar{l}}\right].$$

Using the boundedness of $\{d^i\}$ we conclude that these gradients are Lipschitzian with constant K_1, say. Define now, for $l = 1,\dots,p$, (z^i_l, ν^i_l) by an Armijo step (3.5) along the bounded Frank–Wolfe direction (3.4) applied to the parallel subproblem (3.12) at the point x^i:

$$\begin{aligned}
&\min_{x_l,\lambda_{\bar{l}}}\left\{ f(x_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}}) \;\middle|\; x_l \in X_l,\;\; x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}} \in \prod_{\substack{s=1\\ s\neq l}}^{p} X_s \right\}\\
&\qquad = \min_{x_l,\lambda_{\bar{l}}}\left\{ \theta^i_l(x_l,\lambda_{\bar{l}}) \;\middle|\; x_l \in X_l,\;\; x^i_{\bar{l}} + D^i_{\bar{l}}\lambda_{\bar{l}} \in \prod_{\substack{s=1\\ s\neq l}}^{p} X_s \right\}
\end{aligned} \tag{3.14}$$

By Lemma 3.3 we have that for $l = 1,\dots,p$

$$\theta^i_l(x^i_l, 0) - \theta^i_l(z^i_l, \nu^i_l) \ge \sigma_l(\psi^l(x^i)) \tag{3.15}$$

where $\psi^l(x^i)$ represents the minimum principle optimality function associated with (3.14), and σ_l is the forcing function defined by (3.8). But by the PVD algorithm parallelization step (3.12)

$$\theta^i_l(y^i_l, \lambda^i_{\bar{l}}) \le \theta^i_l(z^i_l, \nu^i_l). \tag{3.16}$$

Hence the last two inequalities yield for $l = 1,\dots,p$

$$\theta^i_l(x^i_l, 0) - \theta^i_l(y^i_l, \lambda^i_{\bar{l}}) \ge \sigma_l(\psi^l(x^i)), \tag{3.17}$$

or

$$f(x^i) - f(y^i_l, x^i_{\bar{l}} + D^i_{\bar{l}}\lambda^i_{\bar{l}}) \ge \sigma_l(\psi^l(x^i)). \tag{3.18}$$

Noting the definition of x^{i_l} in (3.12) we have

$$f(x^i) - f(x^{i_l}) \geq \sigma_l(\psi^l(x^i)), \quad l = 1, \dots, p. \tag{3.19}$$

Thus, for $l = 1, \dots, p$

$$f(x^i) - f(x^{i_l}) \to 0 \implies \sigma_l(\psi^l(x^i)) \to 0.$$

By invoking Lemma 3.5 it follows from the particular form of $\psi^l(x^i)$ for (3.14) that for $l = 1, \dots, p$

$$\sigma_l(\psi^l(x^i)) \to 0 \implies \phi_l(x^i) \to 0,$$

where $\phi_l(x^i) = -\min_{y_l \in X_l} \nabla_l f(x^i)(y_l - x_l^i)$. But since $\sum\limits_{l=1}^{p} \phi_l(x^i) = -\min_{y \in X} \nabla f(x^i)(y - x^i) = \phi(x^i)$, we have that

$$\max_{l=1,\dots,p} \left\{ f(x^i) - f(x^{i_l}) \right\} \to 0 \implies \phi(x^i) \to 0,$$

or equivalently

$$f(x^i) - \min_{l=1,\dots,p} f(x^{i_l}) \to 0 \implies \phi(x^i) \to 0. \tag{3.20}$$

But from the synchronization step (3.13) we have

$$f(x^{i+1}) \leq \min_{l=1,\dots,p} f(x^{i_l}). \tag{3.21}$$

Combining (3.20) and (3.21) gives

$$f(x^i) - f(x^{i+1}) \to 0 \implies \phi(x^i) \to 0 \tag{3.22}$$

as required. □

We now turn our attention to a problem with a more general nonlinear constraint set.

4 PVD with General Constraints

We consider in this section the following problem

$$\min f(x) \text{ such that } g(x) \leq 0 \tag{4.1}$$

where $f\colon \mathbb{R}^n \to \mathbb{R}$ and $g\colon \mathbb{R}^n \to \mathbb{R}^m$ are $C^1(\mathbb{R}^n)$. It turns out that if the constraints are not separable in the sense of Section 3, having a stationary point that results from minimizing the objective function with respect to individual blocks of variables $x_1, \dots, x_p$ and subject to the problem constraints $g(x) \leq 0$ does not results in a useful point as was the case for the unconstrained minimization problem of Section 2, or the separable constraint case of Section 3. This is easily illustrated by the following simple example in $\mathbb{R}^2$:

$$\min x_1^2 + x_2^2 \text{ such that } x_1 + x_2 \geq 2. \tag{4.2}$$

This strongly convex problem has a unique global solution at $\bar{x}_1 = \bar{x}_2 = 1$. However, the point $x_1 = 0.5$, $x_2 = 1.5$ is a global minimum with respect to each of x_1 and x_2 separately, subject to the problem constraint. That is

$$\begin{aligned} 0.5 &= \arg\min_{x_1} \left\{ x_1^2 + 2.25 \mid x_1 \geq 0.5 \right\} \\ 1.5 &= \arg\min_{x_2} \left\{ 0.25 + x_2^2 \mid x_2 \geq 1.5 \right\} \end{aligned}$$

This property is possessed by all points in the first quadrant lying on the constraint $x_1 + x_2 = 2$. None of these points are of any use except the solution $\bar{x}_1 = \bar{x}_2 = 1$. Therefore, it seems that the only sensible way to distribute variables for problems with inseparable constraints is to convert them to unconstrained problems or to problems with separable constraints. Obvious methods for doing so are exterior penalty [8] and augmented Lagrangian methods [16, 3] for handling inseparable constraints while leaving separable constraints as explicit ones. However, a disadvantage of exterior penalty is the unboundedness of the penalty parameter, while the augmented Lagrangian formulation essentially changes the minimization problem into a saddle point problem. An approach that avoids both of these difficulties is the dual differentiable exact penalty function [10] formulation

$$\max_{(x,u)\in \mathbb{R}^{n+m}, u\geq 0} \theta(x,u,\gamma) := L(x,u) - \frac{1}{2}\gamma \left\| \nabla_x L(x,u) \right\|^2 \tag{4.3}$$

where $L(x, u)$ is the standard Lagrangian for (4.1)

$$L(x,u) := f(x) + u^T g(x). \tag{4.4}$$

Various theorems [10] relate (4.3) to (4.1), but the key point is that the penalty parameter γ remains finite and the objective function is differentiable. Thus (4.3) appears to be a reasonable formulation of the problem (4.1) with inseparable constraints as one with separable constraints to which the algorithms of Section 3 are applicable.

Preliminary computational results for the parallel variable distribution algorithm are given in the next section.

5 Computational Results

In this section we report on some preliminary computational results with the PVD algorithm for unconstrained optimization. Our implementation is written in Split–C [6], a parallel extension of the C programming language primarily intended for distributed memory multiprocessors. Split–C is designed around two objectives. The first of these objectives is to capture certain useful elements of shared memory, message passing, and data parallel programming in a familiar context, while eliminating the primary deficiences of each paradigm. The second is to provide efficient access to the underlying machine, which in our work is a Thinking Machines CM-5. In our implementation, shared memory is used to handle data associated with the current best solution and termination conditions. Split–C facilitates easy coding of the synchronization problem which obtains its data via message passing, while allowing the data for the subproblems to be physically distributed across the processors. Much of this can also be carried out using CMMD [17], the message passing library of the CM-5. However, Split–C enables the code to be written in a more readily portable manner.

The current implementation uses MINOS 5.4, a newer version of [13], to solve both the parallel subproblems and the synchronization problem. MINOS uses a Quasi–Newton approach for each of these problems. MINOS was chosen as the optimization tool since it is very reliable, efficient and can be called easily as a subroutine. Furthermore, although other codes may be more efficient for unconstrained optimization (for example, implementations that consider sparse Hessians or approximations), MINOS can also be used to solve large–scale constrained problems. This will enable the unconstrained code used to report results in this section to be generalized for the algorithm given in Section 3.

The test problems used below are a subset of the problems from the constrained and unconstrained testing environment (CUTE) [5]. Both the problems and tools for linking algorithms with the problems are available via anonymous ftp. The problems are written in SIF (standard input format), and include many practical and large–scale examples.

We now give 8 tables of results. The first four tables are on smaller sized problems for which our parallel code could run on one processor. The results reported are for two different choices of directions. In Table 1 and Table 2 the results are obtained using a zero direction, which we will refer to as PVD0. This is closely related to the parallel gradient algorithm [12]. Essentially, this corresponds to a form of the block Jacobi method with synchronization for solving the problems, but is covered by our convergence analysis. Table 1 gives total time in seconds and the total number of function (and gradient) evaluations, while Table 2 reports the speedup efficiency as calculated by

$$\frac{\text{Time on 1 processor}}{(\text{Time on } p \text{ processors}) * p} \tag{5.1}$$

The next two tables, Table 3 and Table 4 give the same results for an implementation using the auxiliary directions d^i generated using a diagonal scaling of the gradient. The diagonal scaling is determined using an extra evaluation of the gradient in each processor to determine an approximate size for the Hessian matrix corresponding to that index. The precise scaling used is the reciprocal of the ith component of the following vector:

$$\nabla f(x + e_i) - \nabla f(x),$$

where x is the current point and e_i represents a vector which has ones in the ith block of variables and zeros elsewhere. We label the corresponding tables with PVD. As above, Table 3 gives the times and function evaluations for PVD, while Table 4 reports the speedup efficiencies as calculated by (5.1) for PVD.

The final four tables report on the same algorithms PVD0 and PVD, but with larger problem sizes. In fact we were unable to run these problem instances with only 1 or 2 blocks due to the lack of sufficient memory on the processors of the CM-5 for running MINOS on the subproblems. Thus Table 5 has results only for 4, 8 16 and 32 processors. The calculations of speedup efficiency in Table 6 are carried out using the following form:

$$\frac{\text{Time on 4 processors} * 4}{(\text{Time on } p \text{ processors}) * p} \tag{5.2}$$

Table 1: PVD0: Time(s) and function evaluations

Problem (Size)	1 proc		2 procs		4 procs		8 procs	
	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)
ARWHEAD(500)	12	(50)	6	(60)	4	(81)	4	(105)
BRYBND(500)	32	(71)	17	(92)	10	(122)	7	(129)
DIXMAANB(300)	5	(52)	4	(65)	2	(87)	2	(132)
DQDRTIC(500)	19	(50)	14	(70)	5	(77)	4	(110)
DQRTIC(500)	270	(533)	120	(801)	69	(1149)	66	(1804)
FREUROTH(500)	1300	(1878)	880	(3627)	480	(4452)	320	(4870)
LIARWHD(500)	28	(76)	8	(77)	8	(121)	9	(200)
MOREBV(100)	63	(1399)	38	(2010)	29	(3199)	31	(6020)
NONDIA(500)	24	(71)	12	(105)	5	(90)	4	(131)
NONDQUAR(500)	29	(63)	16	(100)	9	(140)	5	(163)
PENALTY2(100)	93	(1897)	30	(1128)	19	(1257)	10	(903)
POWER(500)	45	(85)	21	(124)	9	(162)	7	(260)
QUARTC(500)	270	(533)	120	(801)	69	(1149)	66	(1804)
SINQUAD(500)	98	(149)	14	(82)	6	(75)	5	(94)
TOINTGSS(500)	15	(62)	9	(63)	4	(73)	4	(128)
TQUARTIC(500)	44	(95)	6	(57)	4	(60)	2	(68)
VAREIGVL(500)	26	(72)	16	(87)	9	(124)	8	(187)

Table 2: PVD0: speedup efficiencies

Problem (Size)	2 procs	4 procs	8 procs
ARWHEAD(500)	96.8	68.2	37.5
BRYBND(500)	94.1	80.0	57.1
DIXMAANB(300)	72.9	60.7	30.4
DQDRTIC(500)	67.9	99.0	59.4
DQRTIC(500)	112.5	97.8	51.1
FREUROTH(500)	73.9	67.7	50.8
LIARWHD(500)	179.5	87.5	39.8
MOREBV(100)	82.9	54.3	25.4
NONDIA(500)	100.0	127.7	69.8
NONDQUAR(500)	90.6	84.3	67.1
PENALTY2(100)	155.0	122.4	116.2
POWER(500)	107.1	125.0	82.7
QUARTC(500)	112.5	97.8	51.1
SINQUAD(500)	350.0	437.5	235.6
TOINTGSS(500)	85.2	98.7	45.7
TQUARTIC(500)	338.5	297.3	220.0
VAREIGVL(500)	81.3	69.1	41.7

Table 3: PVD: Time(s) and function evaluations

Problem (Size)	1 proc		2 procs		4 procs		8 procs	
	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)
ARWHEAD(500)	12	(50)	6	(62)	4	(77)	4	(97)
BRYBND(500)	33	(71)	25	(119)	11	(131)	11	(207)
DIXMAANB(300)	5	(52)	2	(60)	2	(81)	2	(104)
DQDRTIC(500)	19	(50)	6	(59)	4	(81)	4	(113)
DQRTIC(500)	280	(533)	110	(583)	52	(763)	46	(1198)
FREUROTH(500)	1400	(1878)	170	(462)	740	(6960)	970	(14250)
LIARWHD(500)	29	(76)	45	(141)	75	(680)	6	(145)
MOREBV(100)	65	(1399)	41	(2253)	0	()	0	()
NONDIA(500)	24	(71)	330	(3505)	7	(179)	0	()
NONDQUAR(500)	30	(63)	13	(103)	8	(130)	4	(132)
PENALTY2(100)	96	(1897)	42	(1590)	31	(1908)	20	(1702)
POWER(500)	46	(85)	17	(110)	8	(175)	7	(255)
QUARTC(500)	280	(533)	110	(583)	51	(763)	46	(1198)
SINQUAD(500)	100	(149)	15	(86)	56	(620)	9	(175)
TOINTGSS(500)	15	(62)	7	(60)	4	(73)	5	(120)
TQUARTIC(500)	46	(95)	7	(59)	5	(94)	4	(133)
VAREIGVL(500)	27	(72)	15	(89)	5	(89)	4	(106)

Table 4: PVD: speedup efficiencies

Problem (Size)	2 procs	4 procs	8 procs
ARWHEAD(500)	96.8	68.2	37.5
BRYBND(500)	66.0	75.0	37.5
DIXMAANB(300)	113.0	68.4	34.2
DQDRTIC(500)	150.8	105.6	62.5
DQRTIC(500)	127.3	134.6	76.1
FREUROTH(500)	411.8	47.3	18.0
LIARWHD(500)	32.2	9.7	60.4
MOREBV(100)	79.3	52.4	38.7
NONDIA(500)	2.9	1.8	41.7
NONDQUAR(500)	115.4	92.6	83.3
PENALTY2(100)	114.3	77.4	60.0
POWER(500)	135.3	147.4	79.9
QUARTC(500)	127.3	137.3	76.1
SINQUAD(500)	333.3	44.6	142.0
TOINTGSS(500)	113.6	89.3	36.8
TQUARTIC(500)	328.6	250.0	155.4
VAREIGVL(500)	90.0	125.0	80.4

Table 5: PVD0: Time(s) and function evaluations

Problem (Size)	4 procs		8 procs		16 procs		32 procs	
	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)
ARWHEAD(1000)	14	(85)	11	(120)	9	(158)	10	(253)
BRYBND(1000)	28	(119)	26	(205)	20	(216)	27	(354)
DIXMAANB(1500)	31	(84)	25	(134)	23	(227)	29	(387)
DQDRTIC(1000)	28	(101)	10	(107)	10	(177)	8	(285)
FMINSURF(1024)	20	(102)	16	(139)	19	(239)	18	(283)
MOREBV(1000)	97	(250)	53	(366)	32	(597)	25	(1054)
NONDQUAR(1000)	31	(143)	24	(269)	17	(246)	26	(425)
POWER(1000)	35	(173)	20	(269)	31	(449)	92	(827)
SINQUAD(1000)	19	(83)	13	(93)	10	(116)	11	(163)
TOINTGSS(1000)	18	(90)	10	(115)	11	(222)	11	(379)
TQUARTIC(1000)	14	(68)	8	(75)	6	(99)	5	(146)
VAREIGVL(1000)	32	(135)	20	(184)	22	(249)	32	(394)
WOODS(1000)	20	(126)	14	(159)	15	(237)	41	(397)

Table 6: PVD0: speedup efficiencies

Problem (Size)	8 procs	16 procs	32 procs
ARWHEAD(1000)	63.6	38.9	17.9
BRYBND(1000)	53.8	35.0	13.0
DIXMAANB(1500)	62.0	33.7	13.4
DQDRTIC(1000)	141.4	71.4	41.2
FMINSURF(1024)	62.5	26.3	13.9
MOREBV(1000)	91.5	75.8	48.5
NONDQUAR(1000)	64.6	45.6	14.9
POWER(1000)	87.5	28.2	4.8
SINQUAD(1000)	73.1	47.5	21.6
TOINTGSS(1000)	92.8	40.9	20.5
TQUARTIC(1000)	87.5	63.6	35.0
VAREIGVL(1000)	80.0	36.4	12.5
WOODS(1000)	71.4	33.3	6.1

Finally, Table 7 gives the timings and function evaluations for the larger problems with the auxiliary directions generated by the scaling of the gradient direction. Table 8 gives the corresponding speedup efficiencies for PVD on the larger problems using (5.2).

Table 7: PVD: Time(s) and function evaluations

Problem (Size)	4 procs		8 procs		16 procs		32 procs	
	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)	sec	(fn evals)
ARWHEAD(1000)	15	(80)	8	(92)	9	(139)	9	(216)
BRYBND(1000)	32	(133)	26	(208)	26	(319)	33	(554)
DIXMAANB(1500)	27	(90)	16	(104)	13	(142)	17	(259)
DQDRTIC(1000)	14	(81)	9	(114)	10	(175)	11	(254)
FMINSURF(1024)	21	(100)	16	(135)	13	(172)	25	(300)
MOREBV(1000)	120	(390)	95	(835)	36	(919)	39	(1787)
NONDQUAR(1000)	42	(166)	12	(158)	12	(195)	11	(294)
POWER(1000)	41	(230)	24	(292)	24	(331)	80	(677)
SINQUAD(1000)	160	(623)	30	(247)	22	(271)	32	(449)
TOINTGSS(1000)	12	(76)	11	(122)	11	(181)	10	(282)
TQUARTIC(1000)	15	(89)	12	(146)	9	(209)	10	(351)
VAREIGVL(1000)	27	(118)	11	(116)	14	(181)	19	(342)
WOODS(1000)	41	(184)	12	(136)	16	(211)	29	(381)

We note that there are some very high speedup efficiencies (over 437%) as well as some very low ones (4.8%). We believe that the reason for the low efficiency is that the problems size is not sufficiently large for the number of processors employed. More precisely, in order to attain high speedup efficiency the ratio of the size of the synchronization subproblem to the size of the parallel subproblems (approximately $\frac{p^2}{n}$) must be small. This hypothesis is generally borne out by the numerical results.

Comparing PVD0 and PVD, it is more difficult to say which is the clear winner. For example, comparing the speedup efficiencies for the larger problems (Table 6 and Table 8), 62 % of the efficiencies for PVD are higher that those for PVD0. Thus, there is a slight indication that PVD is more efficient.

We believe that further experimentation on much larger problems with and without constraints is warranted and might reveal a clearer indication of efficiency as well as better algorithmic strategies. These strategies include a variety of different choices for d^i in the parallel subproblems. In addition, asynchronous implementation might overcome some of the low efficiencies obtained by our synchronous algorithm.

Table 8: PVD: speedup efficiencies

Problem (Size)	8 procs	16 procs	32 procs
ARWHEAD(1000)	92.6	43.1	20.8
BRYBND(1000)	61.5	30.8	12.1
DIXMAANB(1500)	84.4	51.9	19.9
DQDRTIC(1000)	75.3	35.7	15.9
FMINSURF(1024)	65.6	40.4	10.5
MOREBV(1000)	63.2	83.3	38.5
NONDQUAR(1000)	175.0	87.5	47.7
POWER(1000)	85.4	42.7	6.4
SINQUAD(1000)	266.7	181.8	62.5
TOINTGSS(1000)	54.5	27.3	15.2
TQUARTIC(1000)	62.5	40.3	18.8
VAREIGVL(1000)	122.7	48.2	17.8
WOODS(1000)	170.8	64.1	17.7

Computational results for multicategory discrimination problems using a closely related algorithm to PVD0 can also be found in [2].

6 Conclusion

Blocks of variables of optimization problems were distributed among parallel processors with each processor taking primary responsibility for updating its assigned block while not forgetting about the other variables by allowing them to vary in a restricted but plausible fashion. A synchronization scheme optimizes over the points obtained by the parallel processors. Convergence results were given for the unconstrained as well as the constrained cases. Preliminary computational results indicate the proposed method has the potential for high parallelization speedup efficiency. An asynchronous implementation, whereby each parallel processor optimizes over the currently available points from the other processors may further increase speedup efficiency. Further theoretical and computational studies are needed to obtain even faster parallel algorithms that take advantage of the powerful multiprocessors that are now available.

Acknowledgement

We are indebted to Ms. Chunhui Chen, a PhD student of one of the authors (OLM) for valuable suggestions regarding the proofs of Theorem 2.1 and Theorem 3.6.

A Appendix

For convenience we state some of the results needed in the paper.

Lemma A.1 *If the nonincreasing sequence of real numbers $\{f_i\}$ has an accumulation point $\bar{f}$ then $\{f_i\}$ converges to $\bar{f}$.*

Proof We first prove that $\{f_i\}$ is bounded below. Let $f_{i_j} \to \bar{f}$. If $\{f_i\}$ is unbounded below then there exist $\bar{i}$, $\bar{j}$ for which we have

$$\begin{array}{rcll} \bar{f} & > & f_{\bar{i}} & \text{(Since } \{f_i\} \text{ is unbounded below)} \\ f_{\bar{i}} & \geq & f_{i_{\bar{j}}} & \text{(Since } \{f_i\} \text{ is nonincreasing)} \\ f_{i_{\bar{j}}} & \geq & \bar{f} & \text{(Since } \{f_{i_j}\} \text{ converges to } \bar{f}\text{)} \end{array}$$

which is a contradiction. Hence $\{f_i\}$ is bounded below and it converges. It must converge to $\bar{f}$ because for a convergent sequence all accumulations points are identical to the limit of that sequence. □

Lemma A.2 ***Quadratic Bound Lemma:*** *[14, p. 144], [4, p. 639] Let $f \in LC^1_K(\mathbb{R}^n)$ then*

$$f(y) - f(x) - \nabla f(x)(y-x) \leq |f(y) - f(x) - \nabla f(x)(y-x)| \leq \frac{K}{2}\|y-x\|^2 \quad \forall x, y \in \mathbb{R}^n.$$

Theorem A.3 ***Linear Convergence Theorem:*** *Let $f\colon \mathbb{R}^n \to \mathbb{R}$, let $\{x^i\} \subseteq \mathbb{R}^n$, let $S = \{x \mid f(x) \leq f(x^0)\}$. If $f \in LC^1_K(S)$ and*

$$f(x^i) - f(x^{i+1}) \geq \alpha \left\|\nabla f(x^i)\right\|^2 \tag{A.1}$$

for some $\alpha > 0$, $i = 0, 1, 2, \ldots$ then every accumulation point $\bar{x}$ of $\{x^i\}$ is stationary, that is $\nabla f(\bar{x}) = 0$.

If in addition, f is strongly convex with parameter $k > 0$, then $\{x^i\}$ converges to the unique solution $\bar{x}$ of $\min_{x\in\mathbb{R}^n} f(x)$ at the linear root rate:

$$\left\|x^i - \bar{x}\right\| \leq \left(\frac{2}{k}(f(x^0) - f(\bar{x}))\right)^{\frac{1}{2}} \left(1 - \frac{2\alpha k^2}{K}\right)^{\frac{i}{2}}.$$

Proof For the first part, let $x^{i_j} \to \bar{x}$. Then

$$f(x^{i_j}) - f(x^{i_{j+1}}) \geq f(x^{i_j}) - f(x^{i_j+1}) \geq \alpha \left\|\nabla f(x^{i_j})\right\|^2 \geq 0.$$

Taking the limit as $j \to \infty$ and invoking the continuity of ∇f we get:

$$0 = f(\bar{x}) - f(\bar{x}) \geq \alpha \left\|\nabla f(\bar{x})\right\|^2 \geq 0.$$

For the second part of the theorem, we note that strong convexity guarantees that S is compact and f is strictly convex. Hence $\{x^i\}$ must have at least one accumulation point $\bar{x}$

which minimizes $f(x)$ on $\mathbb{R}^n$. By the strict convexity of f and the first part of the theorem, every accumulation point must be the same and hence the sequence $\{x^i\}$ converges to $\bar{x}$ which is the unique solution of $\min_{x\in\mathbb{R}^n} f(x)$.

We establish now the linear root rate of convergence of $\{x^i\}$. We have

$$\left\|\nabla f(x^i)\right\| \left\|x^i - \bar{x}\right\| = \left\|\nabla f(x^i) - \nabla f(\bar{x})\right\| \left\|x^i - \bar{x}\right\| \geq (\nabla f(x^i) - \nabla f(\bar{x}))(x^i - \bar{x}) \geq k \left\|x^i - \bar{x}\right\|^2,$$

where the last inequality follows from the strong convexity of f. Hence (A.1) implies

$$f(x^i) - f(x^{i+1}) \geq \alpha k^2 \left\|x^i - \bar{x}\right\|^2.$$

Upon using the Quadratic Bound Lemma A.2 the last inequality yields

$$f(x^i) - f(x^{i+1}) \geq \frac{2\alpha k^2}{K}(f(x^i) - f(\bar{x})).$$

This is equivalent to

$$\left(1 - \frac{2\alpha k^2}{K}\right)(f(x^i) - f(\bar{x})) \geq f(x^{i+1}) - f(\bar{x}).$$

By induction we have then

$$f(x^i) - f(\bar{x}) \leq \left(1 - \frac{2\alpha k^2}{K}\right)^i (f(x^0) - f(\bar{x})). \tag{A.2}$$

Again from strong convexity we have that

$$f(x^i) - f(\bar{x}) \geq \frac{k}{2}\left\|x^i - \bar{x}\right\|^2 + \nabla f(\bar{x})(x^i - \bar{x})$$

or

$$\left(\frac{2}{k}\left(f(x^i) - f(\bar{x})\right)\right)^{\frac{1}{2}} \geq \left\|x^i - \bar{x}\right\|$$

Using the last inequality in (A.2) gives

$$\left\|x^i - \bar{x}\right\| \leq \left(1 - \frac{2\alpha k^2}{K}\right)^{\frac{i}{2}} \left(\frac{2}{k}\left(f(x^0) - f(\bar{x})\right)\right)^{\frac{1}{2}}$$

as required. □

References

[1] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal on Mathematics*, 16:1–3, 1966.

[2] K.P. Bennett and O.L. Mangasarian. Serial and parallel multicategory separation. Computer Sciences Department Technical Report 1165, University of Wisconsin, Madison, Wisconsin, July 1993.

[3] D.P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press, New York, 1982.

[4] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation.* Prentice Hall, Englewood Cliffs, New Jersey, 1989.

[5] I. Bongartz, A.R. Conn, N. Gould, and Ph.L. Toint. Cute: Constrained and unconstrained testing environment. Publications du Départment de Mathématique Report 93/10, Facultés Universitaires De Namur, 1993.

[6] D. Culler. *The Split-C Programming Language.* Computer Science Department, University of California, Berkeley.

[7] M.C. Ferris and O.L. Mangasarian. Minimum principle sufficiency. *Mathematical Programming*, 57(1):1–14, 1992.

[8] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley & Sons, New York, 1968.

[9] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[10] S.-P. Han and O.L. Mangasarian. A dual differentiable exact penalty function. *Mathematical Programming*, 25:293–301, 1983.

[11] O.L. Mangasarian. *Nonlinear Programming.* McGraw–Hill, New York, 1969.

[12] O.L. Mangasarian. Parallel gradient distribution in unconstrained optimization. Technical Report 1145, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706, April 1993.

[13] B.A. Murtagh and M.A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, 1983.

[14] J.M. Ortega. *Numerical Analysis, a Second Course.* Academic Press, 1972.

[15] J.M. Ortega and W.C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables.* Academic Press, 1970.

[16] R.T. Rockafellar. Augmented Lagrange multiplier functions and duality in nonconvex programming. *SIAM Journal on Control and Optimization*, 12:268–285, 1974.

[17] Thinking Machines Corporation, Cambridge, MA. *CMMD Reference Manual, Version 3.0*, 1993.

[18] P. Tseng. Dual coordinate ascent methods for non–strictly convex minimization. *Mathematical Programming*, 59:231–248, 1993.