IBM Research Report

Computational comparisons of some structured trust region approaches to the minimization of nonlinear partially separable functions

Johara Shahabuddin
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016. India.

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

Abstract. Trust region algorithms are strongly convergent, and typically restrict the step to lie within a spherical trust region. Structured trust region algorithms attempt greater efficiency by allowing differing trust region radii in different partially separable subspaces. However, the unpredictable shape of this trust region takes away some convergence strength for naive implementations. Restrictions on the step, as well as new update mechanisms for the trust region radii, have been proposed in earlier work, to correct this.

In the first part of this paper we propose a new structured trust region algorithm to evaluate the practical advantage of structuring the trust region. The new algorithm solves two subproblems in each iteration: a structured set of constraints for one, and a classical spherical constraint in the other. The solution to the structured subproblem is the new step only if it achieves the greater decrease. We prove that this is a first and second order globally convergent strategy.

In the second part we give the results of computational tests on three structured algorithms, two proposed earlier by the author and the one described here, against a typical unstructured trust region algorithm. The structured approaches uniformly do better than the unstructured one.

Key words. trust region algorithm, partial separability, unconstrained, convex constraints, global convergence, structured problem, nonlinear programming, large-scale programming

AMS subject classifications.

90C30, 65K05

1. Introduction. Trust region algorithms minimize a quadratic model of the nonlinear objective function in each iteration, within a region known as the trust region. Typically, the shape of the trust region is spherical, defined by a radius that restricts the step equally in all directions. The size of this radius is updated in each iteration so that that the decrease in the function is accurately modeled by the quadratic. These algorithms have strong convergence properties and can be shown to converge globally under rather generic assumptions even when the quadratic model is only approximately minimized. They are also simple to implement, computationally efficient, and stable.

The minimization problem we are interested in is:

$$(P) \qquad \min_{x \in X} f(x),$$

where X is a closed convex subset of $\Re^n$, and $f : \Re^n \to \Re$.

A nonlinear function is defined to be *partially separable* when it can be written as the sum of p nonlinear *element functions*:

$$f(x) = \sum_{i=1}^{p} f_i(x),$$

where each $f_i(x)$ has a large invariant subspace. Such problems arise frequently in large systems and there has been some work in using their structure to optimize them more efficiently, [3], [11], [14], [19].

Structured trust region algorithms allow the step to have different lengths in the different subspaces, thus allowing longer steps in directions that are more closely approximated by quadratics. Although this approach promises greater efficiency, it is harder now to update the trust region radii in a way so that the decrease in the function is accurately modeled by the quadratic for a computed step. This is because, after a step is taken, the new point may have quite different slope properties from the original point. Thus, when the radii are updated in the conventional and intuitive way, the lengths of the radii no longer reflect the directions where the quadratic terms dominate over higher order terms. An explanation of this is in [12].

In [7] and [12], a restriction on the step is proposed, since their radii update mechanisms cannot ensure convergence. In [13], the gradient and Hessian of the objective function are used to update the trust region radii.

In the first part of this paper we propose a method to evaluate these, and future, structured approaches. We expect a properly designed structured approach to do better than the classical approach, in the sense that between the structured step and the unstructured step, the former should give a better function decrease for a large fraction of the acceptable iterates.

Thus, in the new algorithm, two trust region subproblems are approximately solved in each iteration - one has a single trust region radius which is maintained and updated as in the unstructured trust region algorithm. The other is any structured trust region approach. In each iteration, we first examine a step in the structured trust region (there are no restrictions on this step - it could just as well be a step from a different algorithm altogether). If it satisfies certain conditions, we take the step; and if not, we examine the classical step in the unstructured trust region. Convergence is achieved independent of the structured update mechanism used.

This algorithm shows that it is possible to advantageously exploit the robustness of the single trust region approach, while allowing ourselves the flexibility of a structured one. However, the doubly-constrained algorithm, as we call it, has not been designed for computational efficiency, as we need to solve two subproblems to figure out which gives a better step. The increase in the time per iteration depends on the method used to find approximate solutions to the subproblems.

In the next section we give the algorithm, prove first order convergence for the convex-constrained case, and second order convergence for the unconstrained version of the problem. In the third section we describe the test problems and tools used, the programming of the algorithms, and the results comparing the various algorithms.

2. The Doubly-Constrained Algorithm. The problem (P) is solved iteratively, with $x_0 \in X$ as the given starting point. In each iteration, $f(x_k + s)$, $k = 0, 1, 2, \ldots$ is modeled in terms of the first three terms of its Taylor series. The model is denoted by $m_k(s)$, where s is the step to be solved for. The computed minimizer is s_k, and $x_{k+1} = x_k + s_k$ if the decreases in the function and its model are similar, and the iteration is called *successful*. Else if the decreases in the function and model are dissimilar, $x_{k+1} = x_k$, and the iteration is called *unsuccessful*. The trust region radii are updated, and a new model is generated if required, for the next iteration.

We begin with some basic definitions and assumptions. The l_2-norm is used throughout, unless otherwise specified. (For other norms, the convergence proofs remain valid with changes in values of the appropriate constants.)

A feasibility assumption is made on the convex feasible region X of the minimization problem (P):

Assumption 2.1. *X has a non-empty interior.*

The following assumptions are made about f:

Assumption 2.2. *The function f is bounded below on the set $\mathcal{L} := \{x \in X : \ f(x) \leq f(x_0)\}$.*

Assumption 2.3. *Each f_i, $i = 1, \ldots, p$, and hence f, is twice continuously differentiable on an open set containing $\mathcal{L}$.*

Assumption 2.4. *There exists a constant $\chi \geq 1$ such that $\|\nabla^2 f(x)\| \leq \chi$ on an open set containing $\mathcal{L}$.*

The following notation is used in the course of the argument:

$$\begin{aligned} \delta f_k &= f(x_k) - f(x_k + s_k), \\ \delta m_k &= -m_k(s_k), \\ r_k &= \frac{f(x_k) - f(x_k + s_k)}{-m(s_k)} \\ r_k^a &= \frac{f(x_k) - f(x_k + s_k^a)}{-m(s_k^a)} \\ r_k^b &= \frac{f(x_k) - f(x_k + s_k^b)}{-m(s_k^b)}. \end{aligned}$$

Gradient and criticality measure. We assume that the exact derivative $\nabla f(x_k) = g_k$ is available to simplify our analysis, as discussed in [12]. (The analysis here would continue to hold for another assumption about g_k in [7].)

DEFINITION 2.5. *We define $\alpha(x_k)$ as a* criticality measure *for the problem (P), as follows:*

$$\alpha(x_k) = \alpha_k := | \min_{(x_k + d) \in X} \ \frac{g_k^T d}{\|d\|} \ |. \tag{2.1}$$

Notice that when X is convex, $\alpha_k = 0$ if and only if x_k satisfies first order criticality conditions for the problem (P). (See [7] for a proof of this.) If $X = \Re^n$ (the problem is unconstrained), then it is easy to see that $\alpha_k = \|g_k\|$. We need the following strong assumption on α_k.

Assumption 2.6. *The function $\alpha(x)$ is Lipschitz continuous with the constant L_α.*

Hessian approximation and Rayleigh quotient. Several different assumptions related to the Hessian approximations B_k have been used by earlier authors. We adopt the one used in [7],where it has been explained to be weaker than assuming that B_k is uniformly bounded.

DEFINITION 2.7. *The* generalized Rayleigh quotient *of a function f at x along $s \neq 0$ is defined to be:*

$$\omega(f, x, s) := \frac{2}{\|s\|^2}[f(x + s) - f(x) - \nabla f(x)^T s].$$

Because of the assumption that $\nabla^2 f$ is bounded, we have $|\omega(f_i, x, s)| \leq L_r$ for all i if x and $x + s$ lie in $\mathcal{L}$, where $L_r \geq 1$ is a positive constant. We define a version of the generalized Rayleigh quotient of m_k:

DEFINITION 2.8.

$$\beta_k := 1 + \max_{q=1,\ldots,k} (|\omega(m_q, 0, s_q)|). \tag{2.2}$$

Given these definitions, the assumption is:

Assumption 2.9. $\sum_{k=0}^{\infty} \frac{1}{\beta_k} = +\infty$.

We also make the following assumption, explained in [7], which is needed only for Theorem 2.18. This assumption appears in [8] and [18] as well.

Assumption 2.10. $\lim_{k\to\infty} \beta_k \delta f_k = 0$.

The two subproblems and their trust regions. In the kth iteration, we choose the step in iteration k to be one of the two approximate solutions s_k^a and s_k^b, to the two subproblems (SP_a) and (SP_b).

The trust region radius Δ_k is used in (SP_a), which is the usual unstructured trust region subproblem:

$$\begin{aligned} (SP_a) \qquad \min m_k(s) &= g_k^T s + \frac{1}{2} s^T B_k s, \\ \|s\| &\le \Delta_k, \\ x_k + s &\in X. \end{aligned}$$

The step s_k^a must satisfy a commonly-used sufficient decrease condition, as in [15]:

$$\delta m_k^a = -m_k(s_k^a) \ge \kappa \alpha_k \min(\frac{\alpha_k}{\beta_k}, \Delta_k, 1). \tag{2.3}$$

Such a decrease is achievable. For a proof, see [8].

To define the trust region for (SP_b), we need the following pair of definitions.

DEFINITION 2.11. *The null space N of a function $f(x)$ is defined to be the set $\{v \mid f(x+v) = f(x)\}$.*

DEFINITION 2.12. *The range space R of a function $f(x)$ is defined to be the subspace orthogonal to N in R^n.*

Let R_i denote the range space of an element function $f_i, i = 1, \ldots, p$. Let $\Delta_{i,k}, i = 1, \ldots, p$ be the trust region radii for the p element functions. These radii may be updated in each iteration by comparing the decreases in the element function with the element model decreases. The constraints (SP_b) then define the structured trust region as the intersection of these cylindrical *elemental trust regions.*

$$\begin{aligned} (SP_b) \qquad \min m_k(s) &= g_k^T s + \frac{1}{2} s^T B_k s, \\ \|P_{R_i}(s)\| &\le \Delta_{i,k}, \quad i = 1, \ldots, p, \\ x_k + s &\in X, \end{aligned}$$

where $P_{R_i}(s)$ denotes the projection of a vector s onto R_i. Denote $-m_k(s_k^b)$ by δm_k^b. There is no sufficient decrease condition on δm_k^b required for our convergence results.

2.13. **The doubly-constrained algorithm.** *Given $0 < \mu_1 \le \mu_2 < 1$, a feasible x_0, and starting values for the trust region sizes, the kth iteration takes the following form:*

1. *If the last iteration was successful, calculate g_k and B_k.*
2. *Solve (SP_a) approximately to get s_k^a that satisfies the sufficient decrease condition (2.3).*
3. *Solve (SP_b) approximately to get s_k^b.*
4. *Calculate $\alpha_k^b := \alpha(x_k + s_k^b)$.*
   If $r_k^b \ge \mu_1$, $\delta m_k^b \ge \delta m_k^a$ and

   $$\alpha_k^b - \alpha_k \le \kappa_1 \Delta_k, \tag{2.4}$$

   where κ_1 is a constant, then $s_k = s_k^b$. Else, $s_k = s_k^a$.
5. *Update $\Delta_{i,k}$ by any method. Update Δ_k as follows:*

   $$\begin{aligned} \textit{If } \ r_k^a < \mu_1, \ \Delta_{k+1} &\in [\gamma_1, \gamma_2]\Delta_k, \\ \textit{if } \ r_k^a \ge \mu_2, \ \Delta_{k+1} &\in [1, \gamma_3]\Delta_k, \\ \textit{else, } \ \Delta_{k+1} &= \Delta_k. \end{aligned}$$

6. *If* $r_k \geq \mu_1$, $x_{k+1} = x_k + s_k$. *Else* $x_{k+1} = x_k$.

Some comments on the algorithm are in order:

1. Note that in a successful iteration, either, $\delta f_k = \delta f_k^a \geq \mu_1 \delta m_k^a$, or, $\delta f_k = \delta f_k^b \geq \mu_1 \delta m_k^b \geq \mu_1 \delta m_k^a$. Thus, in either case,

$$\delta f_k \geq \mu_1 \delta m_k^a. \tag{2.5}$$

   We can probably replace the pair of conditions $r_k^b \geq \mu_1$, $\delta m_k^b \geq \delta m_k^a$ by $\delta f_k^b \geq \delta f_k^a$, without affecting convergence. We chose the first set so as to allow a s_k^b to be accepted when $r_k^b \geq \mu_1$, independent of whether $r_k^a \geq \mu_1$ or not.
2. Calculating α_k and α_k^b for step 3 is a difficult problem in general. (It is simple for the unconstrained case where $\alpha_k = \|g_k\|$.) The condition is almost always redundant for a large constant κ_1. Without the condition 2.4 on α_k for the acceptance of s_k^b, we would only get $\liminf_{k\to\infty} \alpha_k = 0$, but not be able to prove $\lim_{k\to\infty} \alpha_k = 0$; Theorem 2.18 depends on this condition.
3. We are free to choose a method of solving for s_k^b, and a way to update the trust region radii for the structured subproblem (SP_b).

The convergence proofs follow those of the shorter-step algorithm in [12], where Δ_k here takes on the role of $\Delta_{\min,k}$ there.

2.1. First order convergence. Here we state and prove first order convergence results for the convex-constrained problem (P). We show that every limit point of the sequence of x_k's generated by the algorithm must be a critical point. We begin with a technical lemma proved in [12], that establishes a lower bound on the accuracy of the model.

LEMMA 2.14. *If Assumption 2.4 holds, then there exists a constant* $L \geq 1$ *such that for each* $k = 0, 1, 2, \ldots$, $|\delta f_k - \delta m_k| \leq L\beta_k \|s_k\|^2$.

LEMMA 2.15. *Consider a sequence of iterates generated by the algorithm and assume that there exists a constant* $\epsilon > 0$ *such that* $\alpha_k \geq \epsilon$ *for all* k. *Then, for sufficiently small* ϵ, $\Delta_k \geq \frac{c_1}{\beta_k}$ *for all* k, *where* $c_1 = \gamma_1 \min(1, \epsilon, \frac{\kappa\epsilon(1-\mu_1)}{L})$.

Proof. The proof is by contradiction. Therefore, assume that Δ_k becomes smaller than $\frac{c_1}{\beta_k}$ for the first time on iteration k. If ϵ is small enough, we can ensure that k is not 0. From the description of the algorithm, this means that in the previous step $r_{k-1}^a < \mu_1$. We try to contradict this, completing the proof.

Now $\Delta_{k-1} \leq \frac{\Delta_k}{\gamma_1} < \frac{c_1}{\beta_k \gamma_1} \leq \frac{\epsilon}{\beta_k} \leq \frac{\epsilon}{\beta_{k-1}}$ since $\{\beta_k\}$ is a non-decreasing sequence. Also, $\Delta_{k-1} \leq \frac{c_1}{\beta_k \gamma_1} \leq \frac{1}{\beta_k} \leq 1$. We substitute this into the sufficient decrease condition to get $\delta m_{k-1}^a \geq \kappa\epsilon\Delta_{k-1}$. Now we have $\frac{|\delta f_{k-1}^a - \delta m_{k-1}^a|}{\delta m_{k-1}^a} \leq \frac{L\beta_{k-1}\Delta_{k-1}^2}{\kappa\epsilon\Delta_{k-1}}$ (from Lemma 2.14) $= \frac{L\beta_{k-1}\Delta_{k-1}}{\kappa\epsilon} \leq \frac{L\beta_{k-1}\Delta_k}{\kappa\epsilon\gamma_1} \leq \frac{Lc_1}{\kappa\epsilon\gamma_1} \leq (1-\mu_1)$ (substituting in the value of c_1). This implies that $r_{k-1}^a \geq \mu_1$. ∎

THEOREM 2.16. *If the sequence of iterates generated by the algorithm is infinite,*

$$\liminf_{k\to\infty} \alpha_k = 0.$$

Proof. Assume, in order to obtain a contradiction, that there exists $\epsilon > 0$ such that $\alpha_k > \epsilon$ for all k, and suppose ϵ is small enough that Lemma 2.15 holds and $\epsilon < 1$. To prove our result we will now try to contradict Assumption 2.9 which states that $\sum_{k=1}^{\infty} \frac{1}{\beta_k} = \infty$ by breaking up the sum over specific subsequences of k. Let S denote the sequence of successful iterations generated by the algorithm. Then $\sum_{k\in S} \delta f_k \geq \mu_1 \sum_{k\in S} \delta m_k^a \geq \mu_1\kappa\epsilon \sum_{k\in S} \min(\frac{\epsilon}{\beta_k}, \Delta_k, 1) \geq \mu_1\kappa\epsilon c_1 \sum_{k\in S} \frac{1}{\beta_k}$, applying the sufficient decrease condition (2.3), (2.5), the result of Lemma 2.15, $c_1 < \epsilon$ and $c_1/\beta_k < 1$. So, from the assumption that $f(x)$ is bounded below, we have that $\sum_{k\in S} \frac{1}{\beta_k} < \infty$.

Now let r be an integer such that $\gamma_3\gamma_2^{r-1} < 1$. Define $n_k = |S \cap \{1, \ldots, k\}|$, the number of successful iterations up to iteration $k \geq 1$. Define $\mathcal{F}_1 = \{k \ : \ k \leq rn_k\}$ and $\mathcal{F}_2 = \{k \ : \ k > rn_k\}$. First we show that $\sum_{k\in\mathcal{F}_1} \frac{1}{\beta_k}$ is finite. If it has only finitely many terms, its convergence is obvious. Otherwise, we may assume that $\mathcal{F}_1$ has an infinite number of elements and then we construct another subsequence $\mathcal{F}_3$ of indices in S in ascending order, with each index repeated r times. Since each $k \in S$ contributes at most r terms, each at least k, to the sequence $\mathcal{F}_1$, the jth term of $\mathcal{F}_3$ is no greater than the jth term of $\mathcal{F}_1$. This, and the monotonicity of the sequence $\{\beta_k\}$, gives us that $\sum_{k\in\mathcal{F}_1} \frac{1}{\beta_k} \leq \sum_{k\in\mathcal{F}_3} \frac{1}{\beta_k} = r \sum_{k\in S} \frac{1}{\beta_k} < \infty$.

Now we show that $\sum_{k\in\mathcal{F}_2}\frac{1}{\beta_k}$ is finite. We can immediately see that $\Delta_k \le \gamma_3^{n_k}\gamma_2^{k-n_k}\Delta_0$. Using Lemma 2.15, we have $\sum_{k\in\mathcal{F}_2}\frac{1}{\beta_k} \le \frac{\Delta_0}{c_1}\sum_{k\in\mathcal{F}_2}(\gamma_3^{n_k}\gamma_2^{k-n_k}) \le$
$\frac{\Delta_0}{c_1}\sum_{k\in\mathcal{F}_2}(\gamma_3\gamma_2^{(r-1)})^{\frac{k}{r}} < \infty$.

Therefore the sum $\sum_{k=0}^{\infty}\frac{1}{\beta_k} = \sum_{k\in\mathcal{F}_1}\frac{1}{\beta_k} + \sum_{k\in\mathcal{F}_2}\frac{1}{\beta_k} < \infty$, which contradicts our assumption. □

Before we prove a theorem claiming $\lim_{k\in S}\alpha_k = 0$, we need the following lemma related to Lemma 2.15:

LEMMA 2.17. *Let $k_1 \ge k$ be the index of the first successful iteration at x_k. Then s_{k_1} achieves a decrease $\delta f_{k_1} \ge \mu_1\kappa\epsilon\min(\frac{c_1}{\beta_{k_1}}, \Delta_k, 1)$, where $\alpha_k = \alpha_{k_1} > \epsilon$ and c_1 is as defined in Lemma 2.15.*

Proof. Case 1. If $\Delta_k < c_1/\beta_k$ then $\Delta_k < \min(1, \alpha_k/\beta_k)$. Hence, $\delta m_k^a \ge \kappa\epsilon\Delta_k$. Thus from Lemma 2.14, using an argument similar to that in Lemma 2.15, we prove that the kth step is a successful one. Thus $k_1 = k$ and the decrease given is achievable by (2.3).

Case 2. If $\Delta_k \ge c_1/\beta_k$ the argument in Lemma 2.15 implies that $\Delta_{k_1} \ge \frac{c_1}{\beta_{k_1}}$. Now by (2.3), once again the decrease is achievable. □

THEOREM 2.18. *If the algorithm generates an infinite sequence S of successful iterates , then* $\lim_{k\in S}\alpha_k = 0$.

Proof. Once again, we prove by contradiction. Assume $\limsup_{k\in S}\alpha_k > \epsilon_1$. From Theorem 2.16, there exists $\epsilon_2 \in (0, \epsilon_1)$ such that there is a subsequence $\{r_j\}$ of successful iterates with $\alpha_{r_j} < \epsilon_2$. Our contradiction assumption then guarantees a subsequence $\{q_j\}$ of successful iterates such that for each j, $\alpha_k \le \epsilon_1$ for $r_j \le k < q_j$ and $\alpha_{q_j} > \epsilon_1$. By renumbering the subsequences if necessary, we suppose $r_{j+1} < q_j$.

For each j, we now find the index p_j, where p_j is set to the first k where $\alpha_k < \epsilon_2$ is encountered while looking at the iterates k in the order $q_j - 1, q_j - 2, \ldots$. We now look at the infinitely many subsequences $K_j = \{k \in S : p_j < k < q_j\}$. Notice that $\alpha_k \ge \epsilon_2$ for all $k \in K_j$, for all j. Let $K = \cup_j K_j$.

Case 1. $\alpha_{p_j+1} < (\epsilon_1 + \epsilon_2)/2$ for infinitely many j's. For each $k \in K$, we have from the sufficient decrease condition, and the fact that all iterations in K are successful, that $\delta f_k \ge \mu_1\kappa\epsilon_2\min(\frac{\epsilon_2}{\beta_k}, \Delta_k, 1)$. Now Assumption 2.10 that $\lim_{\substack{k\to\infty\\ k\in K}}\beta_k\delta f_k = 0$ implies that $\lim_{k\to\infty}\beta_k\Delta_k = 0$, so that for large enough $k \in K$ the minimum above is Δ_k. Hence for large enough $k \in K$, $\delta f_k \ge \mu_1\kappa\epsilon_2\Delta_k$.

If $s_k = s_k^a$, then by Assumption 2.6 about the Lipschitz continuity of α_k we have $\Delta_k \ge |\alpha_{k+1} - \alpha_k|/L_\alpha$. On the other hand, if $s_k = s_k^b$, then from (2.4) we have $\Delta_k \ge (\alpha_{k+1} - \alpha_k)/\kappa_1$. Denote the subsequence of K_j such that $s_k = s_k^a$ by $S_{a,j}$, and define $S_{b,j}$ similarly.

Now from sufficient decrease condition (2.3), the boundedness of f and the fact that iterations in K_j are successful

$$\begin{aligned}
\sum_{k\in K_j}\delta f_k &\ge \mu_1\kappa\epsilon_2\sum_{k\in K_j}\Delta_k \\
&\ge \mu_1\kappa\epsilon_2\Big(\sum_{k\in S_{a,j}}\|s_k\| + \sum_{k\in S_{b,j}}\frac{(\alpha_k^b - \alpha_k)}{\kappa_1}\Big) \\
&\ge \frac{\mu_1\kappa\epsilon_2}{\max(L_\alpha, \kappa_1)}\sum_{k\in K_j}(\alpha_{k+1} - \alpha_k) \\
&\ge \frac{\mu_1\kappa\epsilon_2(\epsilon_1 - \epsilon_2)}{2\max(L_\alpha, \kappa_1)}.
\end{aligned}$$

But this contradicts Assumption 2.2 about the boundedness of f.

Case 2. When Case 1 does not hold, $\alpha_{p_j+1} \ge (\epsilon_1 + \epsilon_2)/2$ for infinitely many j's. We now establish a lower bound on Δ_{p_j+1}. By the same argument as in Case 1, $\Delta_{p_j} \ge \frac{(\alpha_{p_j+1} - \alpha_{p_j})}{\max(L_\alpha, \kappa_1)} \ge \frac{(\epsilon_1 - \epsilon_2)}{2\max(L_\alpha, \kappa_1)}$. So $\Delta_{p_j+1} \ge \frac{(\epsilon_1 - \epsilon_2)}{2\gamma_1\max(L_\alpha, \kappa_1)}$. But then, if k_1 is the first successful iteration after the $(p_j + 1)$th one $(k_1 \le q_j)$ for large enough j we get (from the previous lemma) $\delta f_{k_1} \ge \mu_1\kappa\epsilon_2\Delta_{p_j+1}$ (the β_{k_1} term becomes redundant as in Case 1) $\ge \frac{\mu_1\kappa\epsilon_2(\epsilon_1 - \epsilon 2)}{2\gamma_1\max(L_\alpha, \kappa_1)}$. This contradicts our assumption that f is bounded below. □

THEOREM 2.19. *If the set of successful iterations generated by the algorithm is finite, then all its iterates x_k are equal to some x_* for k large enough, and x_* is critical.*

Proof. From the algorithm, a finite number of successful iterations means that x_k is unchanged for k large enough, and that $x_* = x_j$ where $j-1$ is the index of the last successful iteration. It also means that $\lim_{k\to\infty} \Delta_k = 0$, since for each $k \geq j$, Δ_k is reduced by at least a fraction $\gamma_2 < 1$. Now if $\alpha_j > 0$, we can apply the result of Lemma 2.15 to get a contradiction. Hence $\alpha(x_*) = \alpha_j = 0$, or x_* is critical. ∎

We now go on to show that if (P) is unconstrained, then standard second order convergence results for trust region methods hold.

2.2. Second order convergence in the unconstrained case. We begin by stating the unconstrained problem $(P1)$ about which we prove the results in this subsection:

$$(P1) \qquad \min_{x\in\Re^n} f(x),$$

where $f : \Re^n \to \Re$ is a partially separable function.

The earlier definitions and assumptions apply, by substituting $X = \Re^n$. We strengthen our assumptions about B_k further.

Assumption 2.20. *B_k is the exact Hessian $\nabla^2 f(x_k)$.*

Thus, by Assumption 2.4, we now have $B_k \leq \chi$. Similarly, $\beta_k \leq \chi + 1$, from its definition in (2.2). But we still need the following assumption.

Assumption 2.21. *$\nabla^2 f(x_k)$ is Lipschitz continuous with constant L_c.*

The step must satisfy the sufficient decrease condition below instead of (2.3), since $\alpha_k = \|g_k\|$ for $(P1)$:

$$\delta m_k^a := \delta m_k(s_k^a) \geq \kappa \|g_k\| \min\left(\frac{\|g_k\|}{\beta_k}, \Delta_k, 1\right). \tag{2.6}$$

In addition, the following inequalities must be satisfied when there is a direction of negative curvature. Let λ_k denote the minimum eigenvalue of $\nabla^2 f(x_k)$ as before. Then s_k^a and s_k^b must satisfy

$$\delta m_k^a \geq -\kappa_2 \lambda_k \Delta_k^2, \tag{2.7}$$

$$\delta m_k^b \geq -\kappa_2 \lambda_k \|s_k^b\|^2, \tag{2.8}$$

where κ_2 is a small positive constant. It has been shown in [12] that there exists a step s_k^a simultaneously satisfying conditions (2.6) and (2.7). Thus, there exists a step s_k^b satisfying (2.8).

Finally we state a technical lemma proved in [16].

LEMMA 2.22. *Let x_* be an isolated limit point of a sequence $\{x_k\}$ in $\Re^n$. If $\{x_k\}$ does not converge then there is a subsequence $\{x_{l_j}\}$ of successful iterations which converges to x_* and an $\epsilon > 0$ such that*

$$\|x_{l_j+1} - x_{l_j}\| \geq \epsilon.$$

Now we can prove the first second order convergence result.

THEOREM 2.23. *Let s_k satisfy conditions conditions (2.6), (2.7) and (2.8). If $\{x_k\}$ is the sequence generated by Algorithm 2.13, then the following are true:*

(a) The sequence $\{g_k\}$ converges to zero.

(b) If $\{x_k\}$ is bounded then there is a limit point x_* with $\nabla^2 f(x_*)$ positive semidefinite.

(c) If x_* is an isolated limit point of $\{x_k\}$ then $\nabla^2 f(x_*)$ is positive semidefinite.

(d) If $\nabla^2 f(x_*)$ is nonsingular for some limit point x_* of $\{x_k\}$, then $\nabla^2 f(x_*)$ is positive definite, $\{x_k\}$ converges to x_*, all iterations are eventually successful, and $\{\Delta_k\}$ is bounded away from zero.

Proof.

(a) This is true from the first order convergence theory.

(b) Assume that there is a $\epsilon_1 > 0$ such that for all k large enough, say $k \geq k_0$, $-\lambda_k \geq \epsilon_1$. We will show that this contradicts the assumption that f is bounded. We begin by showing that $\Delta_{k_0} \geq c_2$ for all $k \geq k_0$ (also by contradiction), where $c_2 := \frac{(1-\mu_1)\gamma_1\kappa_2\epsilon_1}{L_c}$. We choose

ϵ_1 to be small enough that $\Delta_{k_0} \geq c_2$. Now suppose $\Delta_k < c_2$ for the first time (for $k > k_0$) on the kth iteration. We have $\Delta_{k-1} \leq \Delta_k/\gamma_1$. From (2.7) and the mean-value theorem,

$$\begin{aligned}\frac{|\delta f^a_{k-1} - \delta m^a_{k-1}|}{\delta m^a_{k-1}} &\leq \frac{\|s^a_{k-1}\|^2 \max_{0\leq\xi\leq 1} \|\nabla^2 f(x_{k-1} + \xi s^a_{k-1}) - \nabla^2 f(x_{k-1})\|}{-\kappa_2\lambda_k\|\Delta_{k-1}\|^2} \\ &\leq \frac{L_c\Delta_{k-1}}{\kappa_2\epsilon_1} \\ &\leq \frac{L_c\Delta_k}{\kappa_2\epsilon_1\gamma_1} \\ &\leq 1 - \mu_1.\end{aligned}$$

Therefore Δ_k could not have been reduced on the $(k-1)$st iteration, or $\Delta_k \geq c_2$ for all k. There must be an infinite number of successful iterations where $r^a_k \geq \mu_1$, since the contrary would lead us to conclude that Δ_k converges to zero. Now $\delta f_k \geq \mu_1\delta m^a_k \geq \mu_1\kappa_2\epsilon_1\Delta^2_k \geq \mu_1\kappa_2\epsilon_1 c_2^2$ for all successful steps, implying that f is unbounded below.

(c) If $\{x_k\}$ converges to x_*, the result follows from (b). Otherwise, Lemma 2.22 applies. Let $\{x_{l_j}\}$ be the subsequence of successful iterations guaranteed there with $\|x_{l_j+1} - x_{l_j}\| \geq \epsilon$, where ϵ is a positive constant. Notice that from (2.7) and (2.8),

$$\delta m_k \geq -\kappa_2\lambda_k\|s_k\|^2.$$

But $\delta f_{l_j} \geq \mu_1\delta m_{l_j} \geq -\kappa_2\mu_1\hat{\lambda}_{l_j}\epsilon^2$, where $\hat{\lambda}_{l_j} = \min(\lambda_{l_j}, 0)$. Since f is bounded below, $\{\hat{\lambda}_{l_j}\}$ must converge to zero and so $\nabla^2 f(x_*)$ is positive semidefinite.

(d) If $\nabla^2 f(x_*)$ is nonsingular for a limit point x_*, then x_* is an isolated limit point by (a). Hence $\nabla^2 f(x_*)$ is positive definite from parts (b) and (c). To prove the rest we go to the following variant of this theorem.

□

THEOREM 2.24. *Let x_k be the sequence generated by the algorithm under the same conditions on the step as in Theorem 2.23. If x_* is a limit point of $\{x_k\}$ with $\nabla^2 f(x_*)$ positive definite then $\{x_k\}$ converges to x_*, all iterations are eventually successful, and $\{\Delta_k\}$ is bounded away from zero.*

Proof. The proof that x_k converges to x_* is the same as that of the corresponding part of Theorem 6.7 in [12]. We repeat it here for ease of reference:

Choose $\epsilon > 0$ and $h > 0$ so that the minimum eigenvalue of $\nabla^2 f(x)$ is at least ϵ for $\|x - x_*\| \leq h$. Since the change in the value of the model δm_k is nonnegative, we have $\|g_k\|\|s_k\| \geq -g_k^T s_k \geq \frac{1}{2}s_k^T\nabla^2 f(x_k)s_k \geq \frac{1}{2}\lambda_k\|s_k\|^2$, where λ_k is the minimum eigenvalue of $\nabla^2 f(x_k)$. Thus $\|x_k - x_*\| \leq h$ implies that

$$\frac{1}{2}\epsilon\|s_k\| \leq \|g_k\|. \tag{2.9}$$

Theorem 2.23 guarantees that $\{g_k\}$ converges to zero, and thus there is an index k_1 for which $\|g_k\| \leq \frac{1}{4}\epsilon h$ for all $k \geq k_1$. Hence, (2.9) shows that if $\|x_k - x_*\| \leq \frac{1}{2}h$ for $k \geq k_1$, then $\|x_{k+1} - x_*\| \leq h$.

Since $g_* = 0$, from the Taylor series expansion of f about x_* we have

$$f(x) - f(x_*) = (x - x_*)^T\nabla^2 f(x_* + \xi x)(x - x_*)/2,$$

where $0 \leq \xi \leq 1$. This implies that for $\frac{1}{2}h < \|x - x_*\| \leq h$, $\nabla^2 f(x_* + \xi x)$ is positive definite and $f(x) - f(x_*) \geq \frac{1}{2}\epsilon\|x - x_*\|^2 > \frac{1}{8}\epsilon h^2$. Thus, there exists an index $k_2 > k_1$ such that $\|x_{k_2} - x_*\| \leq h/2$ and $f(x_{k_2}) \leq f(x_*) + \frac{1}{8}\epsilon h^2$. Applying (2.9) to x_{k_2} and x_{k_2+1}, we get $\|x_{k_2+1} - x_{k_2}\| \leq h/2$. But then $\|x_{k_2+1} - x_*\| \leq h$. Now $f(x_*) + \frac{1}{2}\epsilon\|x_{k_2+1} - x_*\|^2 \leq f_(x_{k_2+1}) \leq f(x_{k_2}) \leq f(x_*) + \frac{1}{8}\epsilon h^2$, implying that $\|x_{k_2+1} - x_*\| \leq h/2$.

Hence, $\|x_k - x_*\| \leq h/2$ for $k \geq k_2$. But since h can be chosen arbitrarily small, $\{x_k\}$ converges to x_*.

To prove that $\{\Delta_k\}$ is bounded away from zero we need to show that r^a_k converges to 1, for which we begin by obtaining a lower bound on δm^a_k.

From (2.9), for k large enough, there exists an ϵ_1 such that $\epsilon_1\|s^a_k\| \leq \|g_k\|$. Thus $\|s^a_k\| \to 0$. We apply this to (2.6) and get $\delta m^a_k \geq \kappa\epsilon_1\|s^a_k\|\min(\frac{\epsilon_1\|s^a_k\|}{\beta_k}, \|s^a_k\|, 1)$. We can choose ϵ_1 so that $\epsilon_1/\beta_k \leq 1$, then for large enough k, $\delta m^a_k \geq \kappa\epsilon_1^2\|s^a_k\|^2/\beta_k \geq \kappa\epsilon_1^2\|s^a_k\|^2/(\chi+1)$. Now $\frac{|\delta f^a_k - \delta m^a_k|}{\delta m^a_k} \leq L_c\|s^a_k\|^3/\delta m^a_k$ (by an argument as in part (b) above) $\leq \frac{L_c(\chi+1)\|s^a_k\|}{\kappa\epsilon_1^2}$, implying that r^a_k converges to 1. Hence $\{\Delta_k\}$ is bounded away from zero and all iterations are eventually successful. □

3. Computational Results. We test our trust region structuring ideas (the shorter-step algorithm in [12], the gradient-dependent algorithm in [13], the doubly-constrained algorithm in the last section) against a comparable version of a typical single trust region algorithm for unconstrained, differentiable test problems. Five trust region radii update mechanisms (from [12]) are also compared. The results are encouraging, and the ideas are shown to be viable within the limitations of the problem set and selected value of unspecified parameters in the algorithms. (The parameter values chosen are based on some preliminary tests.)

Our testing has been done on a single node of the IBM Scalable Power System 2 (SP2) at the Cornell Theory Center. Our algorithms are coded in MATLAB [1], and interfaced with problems from CUTE [2].

The number of iterations, the number of successful iterations, and the time taken to arrive at a local minimum are compared for the different instances. The number of successful iterations equals the number of gradient and Hessian calculations. The three most expensive calculations in our routines are the subproblem solution, updating the structured trust region radii, and gradient and Hessian computations at each new iterate.

3.1. Programming the algorithms. Here we describe the choices made during coding for the various algorithms. We begin with a template used for all the algorithms. Then we give the trust region structure used for the four algorithms. Next is the method of obtaining a solution to the subproblem, followed by fine-tuning of the update mechanisms. Finally, we give some coding details for the structured trust region algorithms.

Exact first and second derivatives are computed. The initial starting value for all trust region radii was chosen to be 1. We (and others, see [10]) found that it takes fewer iterations to reach the solution when we replaced $r_k > \mu_1$ by $r_k > 0$, as the condition for $x_{k+1} = x_k + s_k$.

The single trust region algorithm. The following version of the single trust region algorithm was coded, and used as a template for the other algorithms.

3.1. *The kth iteration is as follows:*

1. *Given* x_k *and* Δ_k*, calculate* g_k *and* $B_k = \nabla^2 f_k$*. Stop if* $\|g_k\| < 10^{-5}$.
2. *Approximately solve subproblem* (SP_a) *(with the feasible region* $X = \Re^n$*) to get* s_k.
3. *Evaluate* $f(x_k + s_k)$*, and hence* r_k.
4. *If* $r_k < .25$ *set* $\Delta_{k+1} = \min(\|s_k\|_\infty, \Delta_k)/2$,
   if $r_k \geq .75$ *and* $\|s_k\|_\infty > \Delta_k/2$*, set* $\Delta_{k+1} = 2\Delta_k$,
   otherwise set $\Delta_{k+1} = \Delta_k$.
5. *If* $r_k > 0$ *set* $x_{k+1} = x_k + s_k$*; else* $x_{k+1} = x_k$.

Trust region structure. As in [13], we define our structured trust region as a hypercube with different bounds for each coordinate. This shape simplifies the solution of the subproblem. In effect, we make the following assumption:

Assumption 3.2. *Each* R_i *is a coordinate subspace, i.e., the span of some set* $e_j, j \in q_i$*, where* $q_i \subset \{1, \ldots, n\}$ *and* e_j *denotes the* j*th unit vector,* $j = 1, \ldots, n$.

Now, each *elemental trust region* is given by the following constraint:

$$\|P_{R_i}(s_k)\|_\infty \leq \Delta_{i,k},$$

where $P_{R_i}(s)$ denotes the projection of a vector s onto R_i. Because we use the ∞-norm, the elemental trust region constraints intersect to give upper and lower bounds on each coordinate.

Thus, the range spaces R_i are enlarged so that they are the span of elementary vectors. (For example, an element function $(x_1+x_2)^3$ that has a range space spanned by $(1,1)'$, is assumed instead to have a range space spanned by $(1,0)'$ and $(0,1)'$.)

At first we solved subproblem (SP_a) (with a ball for its feasible region) for the single trust region algorithm (and (SP_b) (with its feasible region a box) for the structured algorithms). We saw that because of the greater flexibility allowed for the step in the structured trust region (since steps to the corners of a hypercube may be longer by a factor of $\sqrt{n}$ than steps within a sphere enclosed by the box), the single trust region algorithm is at a disadvantage. So we tried three alternatives for both the single and the structured trust regions: the trust region looking like a hypercube, an ellipsoidal trust region to fit inside the hypercube, and with an expansion factor related to $\sqrt{n}$ for the ellipsoidal trust region. We have found that the best option is to have the hypercube structure (by a considerable decrease in the number of iterations) and we chose this for our experiments.

Subproblem solution. We used the same subproblem solution routine for all four algorithms.

If B_k is positive definite and the Newton point lies within the trust region, go to the Newton point. If the Newton point lies outside the trust region, solve the subproblem within the subspace spanned by the Newton direction and the negative gradient direction. Else, if B_k is not positive definite, find a direction of nonpositive curvature. Now solve the subproblem within the subspace spanned by this and the negative gradient direction.

Obtaining an approximate solution to the subproblem by minimization over a two-dimensional subspace spanned by the gradient g_k and a second order direction q_k, which is either the Newton step (when H_k is positive definite) or a direction of nonpositive curvature, is proposed in [5].

We carry out a Cholesky factorization of the Hessian to find q_k. If the factorization is terminated prematurely, we can compute a direction of nonpositive curvature. (This direction might not be a direction of sufficient negative curvature as required for the second order convergence results of the structured methods to hold.) Otherwise, we use the factorization to calculate the Newton direction. All through we ignored the conditions needed for second order convergence, while trying to conform to the conditions for first order convergence. (Trying to implement the former would have meant the solution of a time-consuming eigenvalue problem.)

When the Newton point does not lie within the trust region, the solution to the two-dimensional subproblem must lie within perpendicularity to the negative gradient direction and on the boundary where the trust region intersects the plane. We search for it in the following manner, chosen for its simplicity. The two directions and a vector of coordinate-wise trust region radii (defining a box) are passed to the solution routine, which then tries to minimize the model by efficiently scanning 629 points along the boundary of the two-dimensional subspace intersected by the trust region.

We scan the boundary by dividing up the halfspace within perpendicularity to the negative gradient direction by angle. At each angle to the negative gradient direction from x_k, the distance d to the boundary of the trust region is then computed. The changes in the model value are found for each angle as if each vector to the boundary is considered to be the step. Finally, we pick the vector for which this model value is least, to be the step. The most expensive calculation here is the calculation of the distance to the boundary for each of 629 points, scanning the range $[-1.57, 1.57]$.

We could speed this up by setting to zero some of the coordinates in the directions being scanned that would not affect the distance calculations. This is done by picking a few coordinates and then checking which of the other coordinates are 'dominated' by these in the calculation. The dominated ones are then set to zero.

The above solution routine sometimes gives a negative value for the change in the model δm_k. For example, this may happen where the subspace minimizer is almost orthogonal to the negative gradient direction. In this case, to try to get a positive δm_k we first go to the Cauchy point s_{kc}. If the second order direction (i.e., the Newton or nonpositive curvature direction) further reduces the model value at the Cauchy point we then take an additional step s_1 to the farthest point within the trust region along this direction. (In the case when the second order direction is not the Newton direction, s_k is set to the step that gives the best decrease in the model δm_k from among s_{kc}, s_1, and $s_{kc} + s_1$. We include s_1 in this set since the additional cost of checking it is minor.)

We also tried an exact solution routine for the ellipsoidal version of the subproblems, [15]. The time taken by our method is comparable to the time taken by the exact solution routine, and the number of iterations is far less (due to the shape of the trust region).

Cutting off unsuccessful steps. The reduction of the trust region radius is made stricter for iterations where $r < .25$. To avoid having to take the same step in the next iteration, the trust region is updated to cut off the earlier step. We see this kind of change in the LANCELOT update in [9].

The analog of this step in the structured trust region methods is not as straightforward. The following update for the elemental radii selected for a decrease, may not eliminate the step from the trust region.

$$\Delta_{i,k+1} = \min(\|s_k\|_\infty, \Delta_{i,k})/2.$$

As expected, the effect of implementing this on our initial runs was to have many times more iterations than we got for the following implementation:

$$\Delta_{i,k+1} = \min(\|s_{i,k}\|_\infty, \Delta_{i,k})/2,$$

for all elemental radii selected for a decrease, where the projection of s_k onto R_i is denoted by $s_{i,k}$.. But this turned out to be too restrictive in practice, giving rise to very small trust region radii.

Finally we implemented the above not for all the elemental radii selected for a decrease, but only for the elements for which $\|s_{i,k}\|_\infty$ is larger than $\|\hat{s}_k\|_\infty/2$, where $\hat{s}_k$ is the projection of s_k onto

the subspace spanned by the ranges of all the elements whose radii are selected for a decrease. Our convergence results may not hold for this update.

Limiting the growth of a radius. We limit the growth in the size of the trust region, unless the step length is likely to grow with it. Implementing this change with the structured methods is easy. All elemental radii selected for an increase during the update step, for which the ∞-norm of the elemental step is longer than half the trust region radius are set to twice the ∞-norm of the step. This change does not alter our convergence results.

Values of the constants used in update mechanisms. The constants used for the trust region update mechanisms for the structured algorithms ($\mu_1,\ \mu_2,\ \gamma_1,\ \gamma_2$ and γ_3) have values similar to those used for the single trust region algorithm (given in Algorithm 3.1) for the parallel and combined criteria. For the sloped criteria (where the iteration was still classified by $\mu_1 = .25$ and $\mu_2 = .75$ as in Algorithm 3.1) we computed α_1 and α_2 from $\mu_1 = .35$ and $\mu_2 = .85$ based on our initial experiments with these parameters.

A change in the parallel criterion. A beneficial change in the parallel separation criterion is to calculate π_i, a vector containing the number of variables used by each element and then to change the factor $1/p$ to $\nu_i := \pi_i / \sum_{i=1}^{p} \pi_i$. Since the factor ν_i is a positive constant, the convergence results for all the algorithms still hold. The new form of the criterion is:

3.3 (Changed parallel). *Let* $0 < \mu_1 \leq \mu_2 < 1$, *and let* $\delta f_{i,k} = f_i(x_k) - f_i(x_k + s_k)$, *and* $\delta m_{i,k} = -m_{i,k}(s_k)$.

If $\delta f_{i,k} \geq \delta m_{i,k} - \nu_i(1-\mu_2)\delta m_k$ *then* $\tau^1_{i,k} = 2$,
if $\delta f_{i,k} < \delta m_{i,k} - \nu_i(1-\mu_1)\delta m_k$ *then* $\tau^1_{i,k} = 0$,
else $\tau^1_{i,k} = 1$.

Hybrid trust region. Our initial tests showed that our structured algorithms did slightly better when a *hybrid* method idea from [7], was included. We later found that there were large increases in the number of iterations, for ill-conditioned dense Hessians. To avoid these, we decided not to look at the hybrid method too carefully, at least for now.

The shorter-step method. We relax the shorter-step condition, requiring only that the step be bounded away from orthogonality to the gradient. Since the solution routine scans the range $[-1.57, 1.57]$ instead of scanning the whole solution range $[-\pi/2, \pi/2]$, the relaxed condition is automatically satisfied. When the solution routine gives a negative value of δm_k, and we follow the alternative strategy given above (where we describe the method of obtaining an approximate solution to the subproblem), ignoring the shorter-step condition completely.

The doubly-constrained method. For the doubly-constrained case, we implement the α-condition with $\kappa_1 = 10^{100}$. A departure from Algorithm 2.13 is that instead of $\delta m^b_k > \delta m^a_k$ as part of the condition for testing whether s^b_k is suitable to be a step, in our algorithm we test $\delta f^b_k > \delta f^a_k$. The parallel update criterion is used for the (SP_b) trust region.

The gradient-dependent method. The gradient-dependent method has no conditions on the step. We implemented only the gradient expansion (not the Hessian one) in the trust region radii. If this expansion is done after 'cutting off an unsuccessful step' for iterations where $r < .25$, there is a huge increase in the number of unsuccessful iterations. Hence, the gradient-expansion step is implemented before the unsuccessful step was cut off (which does not conform to the convergence theory in [12]).

Finally a note about how the elemental information is extracted for the gradient-dependent algorithm, and the related conversion of elemental radii into coordinate radii. Making changes in this, speeds up our algorithms by a factor of three. The first method we used for the computations described above was a 'for' loop that calculated the quantity needed element-by-element. This was slow and was replaced by creating a rank-1 matrix from the given vector, and then multiplying it entry-wise by the 0-1 matrix described above. The method that we now use is to create a matrix using MATLAB's 'sparse' function directly from the vectors that contain the nonzero elements of the variable-element matrix (defined below). This matrix is also used in converting the elemental trust region radii $\Delta_{i,k},\ i = 1, \ldots, p$, to bounds on the coordinates of s_k, $\Delta^j_k,\ j = 1, \ldots, n$.

3.2. CUTE examples and tools. The Constrained and Unconstrained Testing Environment [1] (CUTE) is a FORTRAN-based test problem set developed by Conn, Gould and Toint [2] in 1993, whose work on partial separability and trust region methods inspired our algorithms, so we naturally turned to CUTE for test problems.

[1] CUTE has a website - http://www.numerical.rl.ac.uk/cute. It is available free, by anonymous ftp from one of the following sites:
ftp.numerical.rl.ac.uk (internet 130.246.8.22),
thales.math.fundp.ac.be (internet 138.48.20.102).

CUTE is the result of an effort to test LANCELOT, a popular nonlinear optimization code, aimed at large problems. Partial separability is rampant among such problems and the CUTE code reflects the partially separable structure of a problem to efficiently compute the overall function, gradient and Hessian of each test problem. This format is also ideal to get the element-specific information our algorithms use.

Examples. There are some disadvantages to using CUTE. Firstly, the structure of the problem (partial separability, sparsity of the Hessian, diversity of functional forms) is not easily accessible. The second disadvantage is that the unconstrained differentiable test problems in CUTE are all 'academic' rather than 'real', i.e., the test problems are not from actual applications, but from a mathematician's desk.

CUTE stores a vector of variables group-by-group, with another vector that stores pointers to the first variable of each group. (For our problems, we treated the groups as elements.) We use this to generate a sparse matrix representation where we have a matrix with each row corresponding to variable, and each column to an element, obtaining the *variable-element matrix* $V = (v_{ij})$, defined as follows:

$$v_{i,j} = \begin{cases} 1 & \text{if variable } i \text{ belongs to element } j,\ i \in p_j \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

We began selection of problems from CUTE by extracting all the unconstrained, differentiable problems. Several of the above set of problems, involve fewer than six variables and are thus not useful to us. Some of them are quadratic minimization problems for which the change in the function and the model in any step is the same, hence all the algorithms take the same number of iterations to converge. A number of the remaining problems are 'well-behaved' in the same sense — the same number of iterations are taken by all the algorithms for all starting points tried. We also found four unbounded problems and three that are extremely hard to solve. (The number of iterations for some of these exceeded many tens of thousands before I stopped these runs.) None of these are dealt with here.

Among the problems left, eight have a structure that is partially separable (the variable-element matrix has less than 30% nonzero entries, and, so does the Hessian). In the rest, element functions link a large number of the variable so as to negate the assumption of a 'large invariant subspace' (more precisely, either the variable-element matrix or Hessian, or both, have more than 70% nonzero entries). Thus, this set is not partially separable. Although we would like our algorithms to perform as well as the single trust region algorithm on the problems with a dense partial separability structure, we expect improvements in only in the sparse instances.

Of the eight partially separable problems, five allow a variable number of variables. We code each of these with 100, 300, 500 and 1000 variables, to see the effect of problem size on the efficiency of the algorithms. (The same initial letters in the problem name indicate a common original problem from CUTE.)

Tools. On a system where CUTE is installed, to use a test problem one first 'decodes' a particular problem to generate a set of routines that can calculate function and derivative information. At the time that, say, the gradient at a particular point is needed, the request is directed to a decoded gradient calculation tool routine.

There are no existing tools in CUTE to access the element-specific function, gradient and Hessian values needed. So we have created the tools 'nusetup', 'nufn', 'nugr' and 'nuprd' and incorporated them into the CUTE source code. To understand the source code we found the LANCELOT documentation [9] invaluable — the variables used there are similar to the ones used in CUTE.

The program 'nusetup' returns n, the number of variables the problem has, p, the number of *elements*. It also determines l, the number of nonzeros in the $n \times p$ matrix V, defined in (3.1), which tells us which variables used by each group. We need l within 'nusetup' to create the required FORTRAN data structures for the other new tool routines.

The program 'nufn' calculates the values of the element functions (within CUTE this is equivalent to evaluating *group function* values) and returns a vector of these values. The elemental gradients are returned as a sparse array by 'nugr'. We did not need elemental Hessians but only the products of the Hessians with any given vector. The routine 'nuprd' returns these products as a sparse array.

The FORTRAN tool routines had already been interfaced with MATLAB separately by Ingrid Bongartz [2] and by Mary Ann Branch [4] and included with the CUTE software [2]. We extended the interface in [4] so as to connect the new tool routines to MATLAB.

3.3. Results. We begin our discussion of the results from our computations by evaluating the separation criteria, selecting one of them for our further tests. We then examine the performance

TABLE 3.1

Total number of iterations for partially separable problems, using the standard starting point from CUTE, and a hybrid trust region. The letters P, S, and C stand for parallel, sloped and combined criteria. The bottom row of the table gives the total iterations for a similar set of runs without the hybrid trust region.

Problem	*Single*	*Shorter-Step*			*Doubly-Constrained*			*Gradient-Dependent*		
		P	S	C	P	S	C	P	S	C
BROYDN70	63	55	50	65	50	54	58	55	50	65
BROYDN71	158	154	152	164	126	131	132	154	152	164
BROYDN72	218	265	224	276	215	200	208	259	224	276
BROYDN73	445	450	475	486	403	398	410	469	482	472
BRYBND0	18	18	18	18	17	17	17	18	18	18
BRYBND1	29	29	29	29	26	26	27	29	29	29
BRYBND2	22	22	22	22	22	24	24	22	22	22
BRYBND3	31	31	31	31	18	28	28	31	31	31
CHNROSNB	56	56	56	56	54	55	54	56	56	56
ERRINROS	58	62	61	60	51	51	51	63	61	61
EXTROSN0	150	166	154	167	143	116	137	155	147	155
EXTROSN1	151	160	169	160	125	132	155	122	164	134
EXTROSN2	146	125	153	158	164	201	115	125	158	145
EXTROSN3	165	163	151	125	140	165	122	129	125	144
FLETCHC0	191	190	188	188	187	195	187	190	188	188
FLETCHC1	543	536	537	537	536	585	531	536	542	537
FLETCHC2	881	888	888	888	881	1017	885	888	888	888
FLETCHC3	1766	1777	1764	1783	1758	1865	1735	1777	1772	1783
GENROS0	138	128	133	124	98	107	100	129	133	120
GENROS1	366	327	348	329	276	327	266	329	367	330
GENROS2	692	506	593	528	470	597	451	545	577	528
GENROS3	1247	1059	1105	1029	884	1077	853	1056	1131	1059
TOINTPSP	16	16	16	16	14	15	14	16	16	16
Total	7550	7183	7317	7239	6658	7383	6560	7153	7333	7221
No hybrid	-	7303	10533	7286	-	-	-	7288	7492	7369

of the structured algorithms for the partially separable problems. Finally, the algorithms are tested for other classes of unconstrained and differentiable problems in cute: the well-behaved partially separable problems, small problems, quadratic problems and problems with either a dense Hessian, or a dense variable-element matrix, or both.

Performance of separation criteria. The two sloped criteria did not show much difference in their performance, giving identical results for most problems. Neither did the two combined criteria. Table 3.1 contains the results of running the eight partially separable problems for their different sizes with the first sloped and first combined criteria. Here we give the number of iterations taken for the algorithms to converge from the standard starting points of these problems (as available from CUTE). We see that the structured algorithms perform a little bit better (4-5 % fewer iterations) than the single trust region algorithm, specially the doubly-constrained method, for these starting points and problems.

The parallel separation criterion has fewer iterations than the combined criterion, but the difference is not much. The sloped criterion is worse than the other two kinds. The combined criterion tests the ratios between the decreases in the elemental functions and the decreases in the elemental models, and also prevents the radii for elements that have negligible change in the model (compared to the overall change δm_k) from shrinking too much. Thus, we used this criterion in our final comparisons.

The last row of Table 3.1 contains the totals for a similar run where no hybrid trust region is included. We include such a hybrid trust region only for the shorter-step and gradient-dependent algorithms. Notice that it has an effect of decreasing the total number of iterations for the shorter-step algorithm, especially for the sloped criterion.

Performance on partially separable problems. We ran each of the eight partially separable problems (defined earlier as having both a sparse variable-element matrix and a sparse Hessian), with 10 starting points (other than their standard starting point, for which the results are in Table 3.1). Tables 3.2, 3.3 and 3.4 contain the results of this run.

For some of the starting points the number of iterations exceed 4000 (which we thought is a reasonable limit for our set of problems). This happens with starting points that have widely differing values for the different variables. We think this may be due to either the algorithm going into a region where the problem is ill-conditioned (the negative gradient direction has high positive curvature and the second order direction has very slight negative curvature), or the trust region shape becoming skewed so as to limit step length in good directions. (In Table 3.2 the number of such runs have been mentioned in parentheses after the number of iterations.)

We tried to free the data from the effect of these outliers. At first we merely zeroed out, for all the algorithms, all results related to a problem and starting point that 'crashed' (in the sense of hitting 4000 iterations) for any algorithm, so as to completely ignore that particular set of runs. But this biases the results in favour of the algorithms that 'crashed'. So instead we penalise the results for the algorithm, problem and starting point that crashed in the following way: we set the results for these 'crashes' to three times the maximum values reached by other algorithms that had not 'crashed' for the same problem and starting point, for which this maximum is less than 3000. For the instances where the above maximum is greater than 3000, we multiplied by a factor of 1.25 (since we felt that to penalise by more than 1000 iterations would also bias our results). For both of these manipulations, and for various choices of the penalty factors, the results are qualitatively the same.

In Table 3.2, the doubly-constrained and shorter-step methods have fewer iterations on the average than the single trust region method, while the gradient-dependent method has more iterations. In parentheses are the number of starting points for which the run did not converge in 4000 iterations. This happened most often to the gradient-dependent algorithm, which has five such cases, whereas the other three algorithms have at most three each. Also, among the problems, the BRYBND's and FLETCHC's reach the 4000-iteration limit most often. Nine out of the thirteen failures are due to the only starting point with variable values ranging from 1 to 10^6.

In Table 3.3, the total number of successful iterations (where new gradients and Hessians must be computed) for the structured algorithms is always fewer than for the single trust region case.

With a few exceptions, the number of iterations taken by the doubly-constrained method has consistently been found to be less than that taken by the single trust region method. (In the given set of results, only GENROS0 violates this. There are more frequent instances of this Tables 3.6 and 3.7, especially for problems with a dense variable-element matrix.) We must be doing something correctly by having a structured trust region in order to get the consistent decreases we see in the number of iterations.

Of all the problems TOINTPSP has the best results in favour of the structured methods. Its Hessian and variable-element matrix structures have a scattering of off-diagonal elements, whereas all the other problems have a diagonal-heavy structure, with a lot of overlap of variables over any successive pair of elements. Also note that the shorter-step algorithm is remarkably effective for the FLETCHC problems, but not for the others.

Finally, refer to the last line of Table 3.2 to see the effect of a hybrid trust region. (Since some of the starting points are random, the totals in the last and second-to-last rows are not meant directly to be compared with one another, but the relationships of the numbers within the rows are to be compared.) Notice that the relative performance of the shorter-step and gradient-dependent algorithms has become worse vis-a-vis the performance of the single trust region and doubly-constrained methods. This result contradicts our preliminary results in Table 3.1 and the contradiction merits future investigation.

In order to understand our results better we tried to aggregate our numbers in some meaningful ways. In the top part of Table 3.5 we show the behaviour of all the problems that originated from a common CUTE code, but with differing numbers of variables. (The problems are listed with their original names from CUTE.) Here TOINTPSP, CHNROSNB, FLETCHCR give fewer iterations with the structured algorithms than with the single trust region method. We then summed up the number of iterations over all the problems compiled with 100 variables, 300 variables, 500 variables and 1000 variables, separately.

We find that the smallest problems have the fewest iterations for the structured methods. The structured results steadily deteriorate as the size of the problem increases, perhaps because our structured trust region fails to adequately represent nonlinearities when a large number of steps are taken. Each of the problems has very few variables in each element compared with the total number

TABLE 3.2

Comparing the total number of iterations for 10 runs with different starting points, for partially separable problems. The last row has the totals for another set of similar runs with a hybrid trust region. (In parentheses after some of the totals are the number of runs that required 4000 iterations.)

		Total Number of Iterations							
Problem	*n*	*Single*		*Shorter-Step*		*Doubly-Constrained*		*Gradient-Dependent*	
BROYDN70	100	717		797		667		907	
BROYDN71	300	1650		2071		1431		2199	
BROYDN72	500	2399		3045		2093		3388	
BROYDN73	1000	4243		5585		3756		5942	
BRYBND0	100	1801		1554		1176		1547	
BRYBND1	300	3176		2583		1995		3067	(1)
BRYBND2	500	4195		2929		2066		5702	(1)
BRYBND3	1000	4260		4436	(2)	2382		6445	(2)
CHNROSNB	50	805		797		706		736	
ERRINROS	50	1303		1530		1103		1320	
EXTROSN0	100	808		737		745		783	
EXTROSN1	300	847		997		796		1056	
EXTROSN2	500	945		1134		840		1156	
EXTROSN3	1000	949		1482		892		1521	
FLETCHC0	100	3001		1296		2503		2189	
FLETCHC1	300	8095	(1)	2610		6732		4536	
FLETCHC2	500	8783	(1)	5180		8372	(1)	7473	
FLETCHC3	1000	7504	(1)	6423	(1)	6970	(1)	8153	(1)
GENROS0	100	888		924		912		952	
GENROS1	300	2065		2192		1989		2078	
GENROS2	500	3146		3251		3063		3256	
GENROS3	1000	5506		5955		5221		6865	
TOINTPSP	50	565		452		352		430	
Total		67650		57960		56762		71701	
With Hybrid		48002		55334		39377		69745	

of variables (it varies between 1 and 7), and so a hundred variables may already be a large enough size to see the effects of the algorithm. We need to find and test more examples of large partially separable problems to test this hypothesis.

The last two rows of Table 3.5 are the sums for a few of the starting points. The upper row is the sum for starting points 1, 2, 3 for which the total number of iterations with the single trust region method over all problems is the fewest, and the lower one is the sum for starting points 5, 6, 7 for which the above total is the highest (we ignored the starting points where 'crashes' took place). We did this in an attempt to see if the performance of the method has anything to do with the number of iterations needed to solve the problem. However, the entries in these two rows argue against such a correlation.

The variability of these results is in part due to the randomness in solution trajectory inherent in nonlinear problems.

Performance on general problems. Table 3.6 contains results for a few well-behaved partially separable functions, and other general functions selected randomly from among the set of unconstrained differentiable problems from CUTE. The ones with a sparse Hessian are compiled with up to a thousand variables. We found that the problems with dense Hessians not only took a very large amount of time per iteration but also took a very large number of iterations to solve. So we limited ourselves to a hundred variables for these. Results for some quadratic problems and the small problems (fewer than six variables) are presented in Table 3.7. Notice that the structured and single trust region methods do about equally well on these problems in general.

In fact, problems which have dense Hessians but sparse variable-element matrices show fewer iterations for the structured methods over the single trust region method on the average. The

TABLE 3.3
Total number of successful iterations for 10 runs with different starting points for partially separable problems.

	Total Number of Successful Iterations			
Problem	*Single*	*Shorter-Step*	*Doubly-Constrained*	*Gradient-Dependent*
BROYDN70	560	560	578	588
BROYDN71	1297	1392	1258	1272
BROYDN72	1878	1987	1912	2051
BROYDN73	3478	3701	3508	3496
BRYBND0	1200	1239	977	1098
BRYBND1	2122	1998	1687	1777
BRYBND2	2730	2111	1780	2715
BRYBND3	2893	3120	2000	4043
CHNROSNB	609	661	604	573
ERRINROS	963	1050	900	788
EXTROSN0	638	625	635	626
EXTROSN1	660	807	647	761
EXTROSN2	713	952	699	915
EXTROSN3	725	1209	744	1189
FLETCHC0	2550	1056	2121	1786
FLETCHC1	6970	2173	5894	3907
FLETCHC2	7866	4180	7670	6578
FLETCHC3	6223	4938	5975	5616
GENROS0	741	800	812	797
GENROS1	1725	1849	1747	1747
GENROS2	2635	2812	2683	2715
GENROS3	4510	4659	4508	4549
TOINTPSP	456	386	320	345
Total	54142	44265	49659	49932

gradient-dependent algorithm which has more iterations for the partially separable problems, competes well with the other methods here, and sometimes even gives startling reductions in the number of iterations (e.g., FMINSURF in Table 3.6 and BROWNAL in Table 3.7).

4. Conclusions. Out of the four algorithms (the single and three structureds) the one that has the fewest iterations is the doubly-constrained algorithm, which also took the most time. Its performance suggests that there is some value in the idea of structuring. The shorter-step algorithm gives good reductions in the number of iterations on the average. The gradient-dependent algorithm does well for the number of successful iterations, but the gradient-expansion step there is the probable cause for the higher total iterations. With standard starting points, the structured trust region methods give better improvements than with our other ones.

All the separation criteria give convergence in practice although we do not have the theory that shows it. The parallel and combined criteria consistently do better than the sloped one.

The implementation of these algorithms can be improved. Dealing with all the three algorithms, and five update criteria, at once has diluted the amount of attention we could pay to any one method. A number of problems should be looked at individually to try to understand the mechanics of each algorithm. Also, there are a number of simpler strategies whose behaviour can be observed in practice, such as the one described in [12] with an upper bound on ratio of the maximum and minimum of the elemental trust region radii.

We have not implemented any second order conditions in the calculation of the step, nor have we taken advantage of the full flexibility allowed in the trust region radius updating criteria. Many of the parameters involved in checking fits between function and model values are arbitrary. Also, other, faster subproblem solution methods should be considered, such as the dogleg and double dogleg.

TABLE 3.4
Total time taken 10 runs with different starting points for partially separable problems.

	Total Time Taken (in seconds on SP2)			
Problem	*Single*	*Shorter-Step*	*Doubly-Constrained*	*Gradient-Dependent*
BROYDN70	42	67	87	81
BROYDN71	317	447	501	484
BROYDN72	838	1162	1197	1243
BROYDN73	4064	5118	5012	5441
BRYBND0	112	138	158	144
BRYBND1	394	508	562	600
BRYBND2	798	883	893	1668
BRYBND3	1536	2245	1855	3209
CHNROSNB	24	37	47	36
ERRINROS	40	73	78	68
EXTROSN0	34	44	69	50
EXTROSN1	92	145	172	168
EXTROSN2	168	265	286	282
EXTROSN3	317	614	576	672
FLETCHC0	88	71	178	121
FLETCHC1	749	275	854	496
FLETCHC2	1021	780	1513	1222
FLETCHC3	1093	1877	2619	2504
GENROS0	32	53	77	58
GENROS1	115	239	289	248
GENROS2	246	511	634	567
GENROS3	847	1793	2099	2201
TOINTPSP	21	24	29	24
Total	12989	17370	19785	21586

TABLE 3.5
Comparing various aggregates of the total number of iterations for partially separable problems.

	Number of Iterations			
Problem	*Single*	*Shorter-Step*	*Doubly-Constrained*	*Gradient-Dependent*
BROYDN7D	9009	11498	7947	12436
BRYBND	13432	11502	7619	16761
CHNROSNB	805	797	706	736
ERRINROS	1303	1530	1103	1320
EXTROSNB	3549	4350	3273	4516
FLETCHCR	27382	15509	24577	22351
GENROSE	11605	12322	11185	13151
TOINTPSP	565	452	352	430
SIZE(100)	7164	5249	5955	6314
SIZE(300)	11084	10145	9062	11062
SIZE(500)	14391	13585	12256	14722
SIZE(1000)	20842	21560	18711	25682
SUM(1,2,3)	5067	5070	4624	5348
SUM(5,6,7)	27728	26094	23513	29984

TABLE 3.6
Comparing the total number of iterations for general problems.

		Total Number of Iterations			
Problem	n	*Single*	*Shorter-Step*	*Doubly-Constrained*	*Gradient-Dependent*
Easy problems: Sparse Hessians and Sparse Partial Separability					
ARWHEAD	1000	44	44	44	44
BDQRTIC	1000	52	52	52	52
FREUROTH	1000	17	17	17	17
LIARWHD	1000	121	124	90	129
MOREBV	1000	16	16	16	16
NONDIA	1000	33	33	33	33
Total		283	286	252	291
Sparse Hessians and Dense Partial Separability Structure					
DIXMAANA	1500	427	434	144	503
DIXMAANB	1500	683	1932	737	1463
DIXMAANC	1500	1302	1325	785	1467
DIXMAAND	1500	2109	1338	2091	1695
DIXMAANE	1050	979	971	288	627
DIXMAANF	1050	1161	1480	1163	1506
DIXMAANG	1050	1515	1377	2158	1394
DIXMAANH	1050	2269	1889	1854	2170
DIXMAANI	1050	1149	1358	696	1091
Total		11594	12104	9916	11916
Dense Hessians and Sparse Partial Separability Structure					
EIGENALS	110	818	627	431	560
EIGENBLS	110	569	436	487	562
EIGENCLS	132	546	686	441	537
FMINSURF	121	572	136	148	104
PENALTY1	100	176	183	178	176
Total		2681	2068	1685	1939

The types of problems solved so far are quite similar in structure and are set up by other mathematicians. It would useful to tune our algorithms to the solution of two or three 'real' problems. Defining the element functions suitably (since a number of element functions may be grouped together to form a single one, this definition is not unique), and looking at different 'forms' of partial separability would also be useful.

From our observations it seems that for short ranges of steps from the same starting point, the structured trust region algorithms do better. But the longer a sequence of steps is, the higher the probability of the single trust region winning. This can be described in terms of a tortoise and hare story: going slowly, the tortoise adjusts to the environment and makes better progress in the long run; whereas going fast, the hare hits hurdles without expecting them (not being adjusted to the new environment) and is stalled. Perhaps one should look at a method that 'restarts' a structured trust region sequence every few iterations.

Parallelization is an important consideration in the context of large scale nonlinear programming. The doubly-constrained algorithm would certainly speed-up if we used two processors to solve the two subproblems. The three most computationally intensive parts of our programs are amenable to parallelization: (a) the evaluation of elemental functions, gradients and Hessians (b) our subproblem solution routine (where the 600 points to be compared could be distributed between processors) and (c) the trust region updates (each elemental radius is updated independent of the other ones).

The theory needs further study too. It is hard in practice to calculate the criticality measure α_k in the constrained case. A practical method is needed, that does away with this calculation.

5. Acknowledgement. This work was done as part of my Ph.D. at the School of Operations Research and Industrial Engineering at Cornell University, Ithaca, New York. I thank Prof. Michael J. Todd, who was my Ph.D. advisor there.

REFERENCES

[1] *MATLAB User's Guide*, The MathWorks, Inc., Cochituate Place, 24 Prime Park Way, Natick, MA 01760, January 1991, email: info@mathworks.com.

TABLE 3.7
Comparing the total number of iterations for general problems (continued from Table 3.6).

		Total Number of Iterations			
Problem	*n*	*Single*	*Shorter-Step*	*Doubly-Constrained*	*Gradient-Dependent*
Dense Hessians and Dense Partial Separability					
BROWNAL	10	388	398	416	213
MANCINO	20	50	50	50	50
Total		438	448	466	263
Small Problems					
BIGGS6	6	320	331	354	354
BOX3	10	47	47	36	47
BRKMCC	2	9	9	9	9
BROWNBS	2	67	67	65	72
BROWNDEN	4	19	19	19	19
CLIFF	2	15	15	15	15
CUBE	2	135	147	139	152
DENSCHNA	2	36	36	36	36
DENSCHNB	2	17	17	17	17
DENSCHNC	2	29	29	29	29
DENSCHND	3	56	56	54	58
ENGVAL2	3	37	37	37	37
EXPFIT	2	129	130	126	142
GROWTH	3	51	51	51	51
Total		967	991	987	1038
Quadratic Problems					
DIXON3DQ	1000	16	16	16	16
DQDRTIC	1000	16	16	16	16
ENGVAL1	1000	45	45	45	45
HILBERTB	1000	32	32	31	29
Total		109	109	108	106

[2] I. BONGARTZ AND A. R. CONN AND N. GOULD AND PH. L. TOINT, *CUTE: Constrained and Unconstrained Testing Environment*, Technical Report, March 1995, Département de Mathématique, Facultés Universitaires de Namur, Belgium.

[3] A. BOUARICHA, AND J. J. MORÉ, *Impact of partial separability on large-scale optimization*, Comput. Optim. Appl., 7 (1997), pp. 27–40.

[4] M. A. BRANCH, *Getting CUTE with MATLAB*, Technical Report, September 1994, Advanced Computing Research Institute, Cornell University, Ithaca NY 14853, NY, USA.

[5] R. H. BYRD, R. B. SCHNABEL, AND G. A. SHULTZ, *Approximate Solution of the Trust Region Problem by Minimization over Two-Dimensional Subspaces*, Math. Programming, 40 (1988), pp. 247–263.

[6] T. F. COLEMAN, AND Y. LI, *An interior trust region approach for nonlinear minimization subject to bounds*, SIAM J. Optim., 6 (1996), pp. 418–445.

[7] A. R. CONN, N. GOULD, A. SARTENAER, AND PH. L. TOINT, *Convergence properties of minimization algorithms for convex constraints using a structured trust region*, SIAM J. Optim., 6 (1996), pp. 1059–1086.

[8] A. R. CONN, N. GOULD, A. SARTENAER, AND PH. L. TOINT, *Global convergence of a class of trust region algorithms for optimization using inexact projections on convex constraints*, SIAM J. Optim., 3 (1993), pp. 164–221.

[9] A. R. CONN, NICK GOULD, AND PH. L. TOINT, LANCELOT*: A Fortran Package for Large-Scale Nonlinear Optimization*, Springer-Verlag, 1992.

[10] R. FLETCHER, *Practical Methods of Optimization*, Second ed., Practical Methods of Optimization, John Wiley, New York, 1987.

[11] A. GRIEWANK, AND PH. L. TOINT, *On the existence of convex decompositions of partially separable functions*, Math. Programming, 28 (1984), pp. 25–49.

[12] J. SHAHABUDDIN, A shorter-step trust region algorithm for minimization of nonlinear partially separable functions, draft paper, 6 Dec 1999, IBM-India Research Lab, Indian Institute of Technology, Hauz Khas, New Delhi 110 016, India.

[13] J. SHAHABUDDIN, A gradient-dependent trust region algorithm for the minimization of nonlinear partially separable functions, draft paper, 28 Feb 2000, IBM-India Research Lab, Indian Institute of Technology, Hauz Khas, New Delhi 110 016, India.

[14] M. LESCRENIER, *Partially separable optimization and parallel computing*, Ann. Oper. Res., 14 (1988), pp. 213–224.

[15] J. J. MORÉ, *Recent developments in algorithms and software for trust region methods*, in Mathematical Programming: The State of the Art, A. Bachem, M. Grötschel and B. Korte, eds., Springer-Verlag, 1982.

[16] J. J. MORÉ, AND D. SORENSEN, *Computing a trust region step*, SIAM J. Sci. Statist. Comput., 4 (1983), pp. 553–572.

[17] M. J. D. POWELL, *On the global convergence of trust region algorithms for unconstrained minimization*, Math.

Programming, 29 (1984), pp. 297–303.
[18] Ph. L. Toint, *Global convergence of a class of trust region methods for non-convex minimization in Hilbert space*, IMA J. Numer. Anal., 8 (1988), pp. 231–252.
[19] Ph. L. Toint, *Global convergence of the partitioned BFGS algorithm for convex partially separable optimization*, Math. Programming, 36 (1986), pp. 290–306.