Structural Optimization 16, 19–28 © Springer-Verlag 1998

A modified trust region algorithm for hierarchical NLP

S.A. Nelson II and P.Y. Papalambros

Department of Mechanical Engineering and Applied Mechanics, University of Michigan, Ann Arbor, MI 48109, USA

Abstract Large-scale design optimization problems frequently require the exploitation of structure in order to obtain efficient and reliable solutions. Successful algorithms for general nonlinear programming problems with theoretical underpinnings do not usually accommodate any additional structure within the problem. In this article modifications are made to a trust region algorithm to take advantage of hierarchical structure without compromising the theoretical properties of the original algorithm.

1 Introduction

Nonlinear programming (NLP) has now become a common tool in engineering systems design. As problem size and computational expense increase, so does the difficulty of finding solutions. It is generally accepted that in order to find solutions efficiently and reliably, additional structure within the NLP must be exploited (see e.g. Conn *et al.* 1992; Papalambros 1995). The focus here is on modifying existing algorithms in order to accommodate the special type of structure known as *hierarchical decomposition.*

For this work it will suffice to say that a general NLP is hierarchically decomposable if components of the vector of variables $\mathbf{x}$ can be grouped into $p+1$ separate subvectors $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_p$, such that every term in the objective function and every constraint depends only on the *vector of linking variables* $\mathbf{x}_0$ and at most one other vector $\mathbf{x}_j$. That is, the general NLP

$$\min_{\mathbf{x}\in\Re^n} \; f(\mathbf{x}),$$

subject to

$$\mathbf{g}(\mathbf{x}) \le \mathbf{0}, \quad \mathbf{h}(\mathbf{x}) = \mathbf{0}, \tag{1}$$

can be cast in the form

$$\min_{\mathbf{x}\in\Re^n} \; f_0(\mathbf{x}_0) + \sum_{j=1}^{p} f_j(\mathbf{x}_0, \mathbf{x}_j),$$

subject to

$$\mathbf{g}_0(\mathbf{x}_0) \le \mathbf{0}, \quad \mathbf{h}_0(\mathbf{x}_0) = \mathbf{0},$$

$$\mathbf{g}_j(\mathbf{x}_0, \mathbf{x}_j) \le \mathbf{0}, \; j = 1, \ldots, p,$$

$$\mathbf{h}_j(\mathbf{x}_0, \mathbf{x}_j) = \mathbf{0}, \; j = 1, \ldots, p, \tag{2}$$

because $\mathbf{g}^T = [\mathbf{g}_0^T, \mathbf{g}_1^T, \mathbf{g}_2^T, \ldots, \mathbf{g}_p^T]$ and $\mathbf{h}^T = [\mathbf{h}_0^T, \mathbf{h}_1^T, \mathbf{h}_2^T, \ldots, \mathbf{h}_p^T]$. For the same reason, there is no ambiguity between writing $\mathbf{h}_j(\mathbf{x})$ and $\mathbf{h}_j(\mathbf{x}_0, \mathbf{x}_j)$ because $\mathbf{h}_j$ depends only on those elements within $\mathbf{x}$ which correspond to the subvectors $\mathbf{x}_0$ and $\mathbf{x}_j$. Optimization formulations of engineering system design problems often have this hierarchically decomposable structure. Rigorous methods for identifying a hierarchically decomposed structure within an existing NLP can be found in the paper by Michelena and Papalambros (1995) and for formulating an NLP such that hierarchy is prevalent in that of Krishnamachari and Papalambros (1997).

Solving an approximation of (1) can be expensive if the number of variables or constraints is large, and inaccurate if second-order information is approximated. Therefore, the decomposed form (2) is used to define several *subproblems*, each subproblem being the collection of functions which depend on a particular subvector of variables $\mathbf{x}_j$ for $j = 1, 2, \ldots, p$ in the form

$$\min_{\mathbf{x}_j\in\Re^{n_j}} \; f_j(\mathbf{x}_0, \mathbf{x}_j),$$

subject to

$$\mathbf{g}_j(\mathbf{x}_0, \mathbf{x}_j) \le \mathbf{0}, \quad \mathbf{h}_j(\mathbf{x}_0, \mathbf{x}_j) = \mathbf{0}. \tag{3}$$

Each subproblem (3) is necessarily smaller than the undecomposed problem (2) and can be solved independently if the linking variables $\mathbf{x}_0$ are treated as parameters.

In a typical hierarchical framework a *master problem* (sometimes called a *coordination problem*) is solved in terms of $\mathbf{x}_0^*$ for given $\mathbf{x}_j$'s. The predicted value for the optimal $\mathbf{x}_0^*$ is passed to the subproblems, each of which is solved with respect to $\mathbf{x}_j$. The values at $\mathbf{x}_j^*$ are returned to the master problem and the process is repeated until some convergence criterion is satisfied. Due to desirable properties such as parallelism and subproblem autonomy, hierarchical frameworks have been studied extensively in engineering design (see Sobieski 1987; Thareja and Haftka 1990). The success of NLP algorithms is at least in part due to established theoretical properties (global and local convergence), yet many hierarchical NLP algorithms in the engineering literature tend to have an ad hoc nature. For this reason, the work presented here focuses on modifying existing algorithms and retaining their theoretical properties as opposed to developing new algorithms.

Guidelines originally proposed by Nelson and Papalambros (1997) can be used to modify an established algorithm to work in a hierarchical manner. The modified algorithm

acquires the advantages of a hierarchically structured algorithm, namely, the utilization of smaller and more accurate approximations, parallelism, and subproblem autonomy without compromising the convergence properties of the original algorithm.

In Section 2, an overview of the modifications is given, followed by an explanation of a trust region algorithm (TR) by Yuan (1995) in Section 3. In Section 4, the guidelines from Section 2 are applied to Yuan's algorithm, resulting in the sequentially decomposed trust region (SDTR) algorithm. In Section 5, two numerical examples are given for which both the TR and SDTR algorithms are used to find a solution, followed by the conclusions in Section 6.

2 Guidelines for modifying NLP algorithms

A general NLP algorithm can be summarized by the following simple steps.

Algorithm 1. General outline of a typical NLP algorithm

1. Start with some initial point $\mathbf{x}^1$, an iteration counter $k = 1$, and any other necessary parameters.
2. If the convergence criteria are satisfied then stop. Otherwise at $\mathbf{x}^k$, solve an approximation to the NLP (1), giving a direction for improvement $\mathbf{s}^k$.
3. If the candidate point $\mathbf{x}^k+\mathbf{s}^k$ is acceptable, define $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k+1$, and go to Step 2.
4. Otherwise, take some action to restrict $\mathbf{s}^k$ (i.e. either perform a line search or reduce the trust region) until $\mathbf{x}^k+\mathbf{s}^k$, is acceptable. Then define $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k+1$, and go to Step 2.

To accommodate the hierarchy in (2), a step is added during which the linking variables $\mathbf{x}_0$ are held at their current value, and each subproblem is optimized while treating the linking variables as parameters. Nelson and Papalambros (1997) have argued that care must be taken with this extra step in order to retain the properties of the original algorithm. The resulting guidelines are summarized here.

First, to retain the global convergence properties of the original algorithm, coordination between subproblems is performed using an approximate problem which has the same form as the unmodified algorithm. For example, a quadratic program would be used if the guidelines were applied to an SQP algorithm.

Second, in order to retain any established local convergence properties the subproblems are not solved independently when near a solution, i.e. the additional step is not used.

Third, the subproblems are used not only to improve their respective objective functions while maintaining or obtaining feasibility, but also to give better estimates of other quantities used in the algorithm, such as penalty parameters, Hessian estimates, and trust region radii.

Any algorithm which follows these guidelines is referred to as a sequentially decomposed (SD) algorithm. The general outline of the SD algorithm (Algorithm 2) differs very little from the original algorithm (Algorithm 1).

Algorithm 2. General outline of a sequentially decomposed (SD) algorithm

1. Start with some initial point $\mathbf{x}^1$, an iteration counter $k = 1$, and any other necessary parameters.
2. If the current iterate $\mathbf{x}^k$ is not near the solution, then temporarily treat the linking variables $\mathbf{x}_0$ as parameters and solve each subproblem (3) to obtain a new value for $\mathbf{x}_j$, $j = 1,2,\ldots,p$. If the current iterate $\mathbf{x}^k$ is near the solution, go to Step 3.
3. If the convergence criteria are satisfied then stop. Otherwise solve an approximation to the NLP (1) at $\mathbf{x}^k$ giving a direction $\mathbf{s}^k$.
4. If the candidate point $\mathbf{x}^k+\mathbf{s}^k$ is acceptable, define $\mathbf{x}^{k+1} = \mathbf{x}^k+\mathbf{s}^k$, $k = k+1$, and go to Step 2. Otherwise, take some action to restrict $\mathbf{s}^k$ (i.e. either perform a line search or reduce the trust region) until $\mathbf{x}^k+\mathbf{s}^k$ is acceptable. Then define $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k+1$, and go to Step 2.

Intuitively, the changes implied by these guidelines would seem to make an algorithm run more efficiently by taking advantage of some of the structure within the problem. The proofs that the modified algorithms retain the convergence properties of the original algorithms are given in the dissertation by Nelson (1997), but these proofs do not guarantee that the modified algorithm is more efficient in any way. Computational gains, if any, will become evident only through experimentation as shown in Section 5. In the next two sections, the above guidelines are applied to a trust region algorithm.

3 A trust region algorithm

In a trust region algorithm, candidate steps are calculated by solving a simple approximate problem, but each step is contained within a finite region where it is assumed the approximate model is reasonably accurate.

If the candidate step gives a good prediction for the changes in the objective and the constraints, the size of the region is increased. If the candidate step does not produce the expected changes the region is made smaller in order to ensure convergence. The basic steps of a trust region algorithm are given in Algorithm 3.

Algorithm 3. Outline of a basic trust region (TR) algorithm

1. Set the iteration counter $k = 1$. Choose some initial point $\mathbf{x}^1$, an initial Hessian estimate $\mathbf{B}^1$, and a trust region radius $\Delta^1 > 0$.
2. Calculate a candidate step by solving an appropriate approximate problem while keeping the step s bound within the trust region Δ. Denote the solution to the approximate problem $\mathbf{s}^k$. If $\mathbf{s}^k = \mathbf{0}$ then stop.

3. Decide if the candidate point $\mathbf{x}^k + \mathbf{s}^k$ is acceptable by calculating a penalty function $\Phi(\mathbf{x}^k + \mathbf{s}^k)$, an approximate penalty function $\phi(\mathbf{s}^k)$, and comparing the two quantities. If the candidate point $\mathbf{x}^k + \mathbf{s}^k$ is not acceptable, then reduce the size of the trust region and go to Step 2, otherwise proceed to Step 4.

4. Adjust the trust region radius according to how well the approximate penalty function ϕ predicts the change in the actual penalty function Φ.

5. Generate the next Hessian estimate $\mathbf{B}^{k+1}$.

6. Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k + 1$ and go to Step 2.

The theoretical framework of trust region algorithms keeps steps bound, thus providing for two distinct advantages over line search methods. First, a trust region algorithm can utilize negative curvature. Methods such as sequential quadratic programming (SQP) force the Hessian approximation to be positive definite in order to guarantee a solution for the quadratic program, so if the actual problem has negative curvature the algorithm cannot model it. In trust region algorithms, no such restriction is made.

The second advantage is embedded in the manner that second-order information is used. If second-order estimates are bad, line search algorithms will use subsequently erroneous search directions, resulting in only small movement and excessive function calls or even failure during the line search. In contrast, a trust region algorithm will reduce the domain over which the approximate model is believed accurate, effectively decreasing the detrimental effects of bad second-order estimates.

However, there are also disadvantages to trust region algorithms. First, when the trust region is too large, the approximate problem is usually re-solved after reducing the trust region, which is more computationally expensive than a line search. Additionally, the approximate problems are in general more difficult to solve because of the possible indefiniteness of the Hessian.

Yuan's algorithm (1995) uses the nonsmooth penalty function given in (3)

$$\Phi(\mathbf{x}) = f(\mathbf{x}) + \sigma \|\mathbf{c}(\mathbf{x})^+\|_\infty . \tag{4}$$

To reduce notation, the vectors of inequalities $\mathbf{g}$ and equalities $\mathbf{h}$ have been concatenated into a single vector $\mathbf{c}^T = [\mathbf{g}^T, \mathbf{h}^T]$. The σ is a penalty parameter and the superscript $+$ denotes the amount of violation of a constraint. If the element c_i corresponds to an equality constraint, then $c_i^+ = |c_i|$, and if c_i corresponds to an inequality constraint, then $c_i^+ = \max(0, c_i)$

$$\|\mathbf{c}(\mathbf{x})^+\|_\infty =$$

$$\max\left(0, \max_{i=1\ldots m_{\text{ineq}}}\{g_i\}, \max_{i=1\ldots m_{\text{eq}}}\{|h_i|\}\right) . \tag{5}$$

At each iteration, a candidate step is calculated by minimizing the approximate problem

$$\min_{\mathbf{s}\in\Re^n} \quad \phi(\mathbf{s}) = \nabla f \mathbf{s} + \frac{1}{2}\mathbf{s}^T\mathbf{B}\mathbf{s} + \sigma\|(\nabla \mathbf{c}\mathbf{s} + \mathbf{c})^+\|_\infty ,$$

subject to

$$\|\mathbf{s}\|_\infty \le \Delta . \tag{6}$$

Any candidate step produced by (3) is bound within a box-shaped region of radius Δ. A step is accepted if it produces improvement in the penalty function, i.e. if $\Phi(\mathbf{x}) - \Phi(\mathbf{x}+\mathbf{s}) > 0$. The trust region radius is altered at each iteration according to how accurately the approximate function ϕ predicts changes in the actual penalty function Φ, using the rule

$$\Delta^{k+1} = \begin{cases} \max(2\Delta, 4\|\mathbf{s}\|_\infty) & \text{if } 0.9 < r \\ \Delta^k & \text{if } 0.1 \le r \le 0.9 \\ \min(\Delta/4, \|\mathbf{s}\|_\infty/2) & \text{if } r < 0.1 \end{cases} ,$$

$$\text{where } r = \frac{\Phi(\mathbf{x}) - \Phi(\mathbf{x}+\mathbf{s})}{\phi(\mathbf{0}) - \phi(\mathbf{s})} . \tag{7}$$

Thus, if ϕ predicts the behaviour of Φ well, the trust region radius is expanded so that future iterations can move more aggressively toward a solution $\mathbf{x}^*$. However, if ϕ does not predict the behaviour of Φ well, the trust region is reduced.

In order to guarantee convergence for trust region algorithms each iteration must satisfy the fractional Cauchy descent property originally established by Powell (1975). For the unconstrained case, there exists some constant γ such that

$$\phi(\mathbf{0}) - \phi(\mathbf{s}^k) < \gamma\|\nabla f(\mathbf{x}^k)\| \min\left(\Delta^k, \frac{\|\nabla f(\mathbf{x}^k)\|}{\|\mathbf{B}^k\|}\right) . \tag{8}$$

Yuan uses the parameter δ^k as an estimate for γ and the related condition

$$\phi(\mathbf{0}) - \phi(\mathbf{s}^k) < \delta^k\sigma^k \min(\Delta^k, \|\mathbf{c}(\mathbf{x}^k)^+\|_\infty) , \tag{9}$$

for the constrained case in order to force convergence and update the penalty parameter. Basically, if (3) does not hold for some iteration, then the penalty parameter σ is increased and the parameter δ is decreased for all future iterations. For more details, the reader is referred to Yuan (1995).

4 The SDTR algorithm

The modified sequentially decomposed trust region (SDTR) algorithm is as follows.

Algorithm 4. The SDTR algorithm

1. Set the outer iteration counter $k = 1$. Choose some $\mathbf{x}^1$ as an initial point and a set of $\mathbf{B}_j^1$ for $j = 0, 1, 2, \ldots, p$ as initial sparse Hessian estimates for each subproblem. Also define some Cauchy parameter estimate $\delta^1 > 0$, penalty parameter $\sigma^1 > 0$, and trust region radius $\Delta^1 > 0$.

2. If the current iterate $\mathbf{x}^k$ is near the solution then proceed to Step 3, otherwise solve each subproblem using the current value for the penalty parameter σ^k, trust region radius Δ^k, and Cauchy parameter estimate δ^k by applying Yuan's algorithm to each subproblem (3) for $j = 1 \ldots p$. Upon completion, set the parameters for the coordination

problem by defining $\Delta^k = \min_j(\Delta_j^\dagger)$, $\sigma^k = \max_j(\sigma_j^\dagger)$, $\delta^k = \max_j(\delta_j^\dagger)$, and $\mathbf{x}^k = [\mathbf{x}_0^k, \mathbf{x}_0^\dagger, \mathbf{x}_1^\dagger, \ldots, \mathbf{x}_p^\dagger]$ where the superscript $\dagger$ is used to denote values upon completion of each subproblem.

3. Using $\mathbf{B} = \sum_{j=0}^{p} \mathbf{B}_j^k$ solve the approximate problem (3) and denote the solution $\mathbf{s}^k$. If $\mathbf{s^k} = \mathbf{0}$ then stop.
4. Calculate the penalty function Φ, approximate penalty function ϕ and the ratio r. If $r > 0$ go to Step 5, otherwise set $\Delta^k = \|\mathbf{s^k}\|_\infty/4$, $\mathbf{x}^{k+1} = \mathbf{x}^k$, $k = k+1$ and go to Step 3.
5. Alter the trust region radius according to (7).
6. Generate $\mathbf{B}_j^{k+1}$ for each $j = 0, 1, 2, \ldots, p$.
7. If $\phi(\mathbf{0}) - \phi(\mathbf{s}^k) < \delta^k \sigma^k \min(\Delta^k, \|\mathbf{c}(\mathbf{x})^+\|_\infty)$ then set $\sigma^{k+1} = 2\sigma^k$ and $\delta^{k+1} = \delta^k/4$, otherwise set $\sigma^{k+1} = \sigma^k$ and $\delta^{k+1} = \delta^k$.
8. Set $\mathbf{x}^{k+1} = \mathbf{x}^k + \mathbf{s}^k$, $k = k+1$ and go to Step 2.

Note that if Step 2 were omitted from the modified algorithm, the remaining steps could be viewed as a version of Yuan's original algorithm. What is meant in Step 2 by "apply Yuan's algorithm to each subproblem" is that the variables $\mathbf{x}_0$ are held constant and Yuan's original algorithm is used to find a solution to *each of the subproblems* (3). Also in Step 2, the starting values of the parameters Δ, σ, δ and the Hessian estimate $\mathbf{B}_j^k$ for the subproblems are the current values used by the SDTR algorithm. The SDTR algorithm actually uses p separate Hessian estimates instead of just one. Yuan's algorithm and analysis are general enough to include this possibility. Upon completion of Step 2, the penalty parameter, the trust region radius, and the Cauchy parameter have all been altered – presumably to better estimates.

4.1 Updating sparse Hessian estimates

According to the definition of a subproblem (3), every function in a subproblem depends only upon the two subvectors $\mathbf{x}_0$ and $\mathbf{x}_j$. Furthermore, the third guideline in Section 2 states that subproblems should be used not only to improve their respective objective functions while maintaining or obtaining feasibility, but also to give better estimates of other quantities used in the algorithm such as penalty parameters, Hessian estimates, and trust region radii.

By rearranging and grouping the terms of the Lagrangian function

$$L(\mathbf{x}, \mu, \lambda) = f(\mathbf{x}) + \mu \mathbf{g}(\mathbf{x}) + \lambda \mathbf{h}(\mathbf{x}) =$$

$$f_0(\mathbf{x}_0) + \sum_{j=1}^{p} f_j(\mathbf{x}_0, \mathbf{x}_j) + \mu_0 g_0(\mathbf{x}_0) + \mu_j g_j(\mathbf{x}_0, \mathbf{x}_j),$$

$$\lambda_0 h_0(\mathbf{x}_0) + \lambda_j h_j(\mathbf{x}_0, \mathbf{x}_j),$$

$$L_0(\mathbf{x}_0, \mu_0, \lambda_0) + \sum_{j=1}^{p} L_j(\mathbf{x}_0, \mathbf{x}_j, \mu_j, \lambda_j), \qquad (10)$$

where μ and λ are row vectors of Lagrange multipliers for the inequality and equality constraints, and μ_j and λ_j are the appropriate vectors of Lagrange multipliers for the j-th subproblem, a term $L_j(\mathbf{x}_0, \mathbf{x}_j, \mu_j, \lambda_j)$ for each subproblem can be defined. In a manner similar to Griewank and Toint (1982b,a) and Griewank (1991), SD algorithms use a Hessian estimate for each subproblem, or equivalently, a Hessian estimate $\mathbf{B}_j$ is used to approximate each term $\nabla^2 L_j(\mathbf{x}_0, \mathbf{x}_j, \mu_j, \lambda_j)$. Each Hessian is of dimension $n \times n$, but has zero values for any element that does not correspond to $\mathbf{x}_0$ or $\mathbf{x}_j$. Updates are made to each Hessian estimate by defining a step $\bar{\mathbf{s}}_j \in \Re^n$ which has zero values for every element except for those that correspond to $\mathbf{x}_0$ and $\mathbf{x}_j$

$$\bar{\mathbf{s}}_j^T = \left[\mathbf{x}_0^{k+1} - \mathbf{x}_0^k, 0, \ldots, 0, \mathbf{x}_j^{k+1} - \mathbf{x}_j^k, 0, \ldots, 0,\right]. \qquad (11)$$

The change in the gradient of the Lagrangian of a subproblem after taking the step $\bar{\mathbf{s}}_j$ is then

$$\mathbf{y}_j = \nabla L_j(\mathbf{x}_0^{k+1}, \mathbf{x}_j^{k+1}, \mu_j, \lambda_j) - \nabla L_j(\mathbf{x}_0^k, \mathbf{x}_j^k, \mu_j, \lambda_j) =$$

$$\nabla L_j(\mathbf{x}^k + \bar{\mathbf{s}}_j, \mu_j, \lambda_j) - \nabla L_j(\mathbf{x}^k, \mu_j, \lambda_j). \qquad (12)$$

Note that the row vector $\mathbf{y}_j$ also has zero values for any elements which do not correspond to $\mathbf{x}_0$ or $\mathbf{x}_j$, as do the rank one matrices $\bar{\mathbf{s}}_j \bar{\mathbf{s}}_j^T$, $\mathbf{y}_j^T \bar{\mathbf{s}}_j^T$, and $\mathbf{y}_j^T \mathbf{y}_j$.

Using these definitions for $\bar{\mathbf{s}}_j$ and $\mathbf{y}_j$, the sparse Hessian estimate $\mathbf{B}_j$ can be updated using any of the popular formulae (i.e. BFGS, DFP, PSB, or any of the Broyden class), and $\mathbf{B}_j^{k+1}$ will be an effective sparse estimate of the Hessian for the j-th subproblem. This technique is also used during the subproblems, so that the whole Hessian is updated even though only the elements corresponding to $\mathbf{x}_j$ are needed by the subproblem.

4.2 Testing for closeness

Step 2 in the SDTR algorithm does not specify how to test when the current iterate $\mathbf{x}^k$ is close to the solution $\mathbf{x}^*$. This test is needed to preserve the local convergence properties of the original algorithm, but if the test is restrictive and each subproblem is solved when $\mathbf{x}^k$ is close to $\mathbf{x}^*$, the SD algorithm may not converge at the expected rate. However, if the decision is made aggressively, so that the subproblems are not used when $\mathbf{x}^k$ is relatively far from the solution $\mathbf{x}^*$, then computational benefits may not be realized.

Because Yuan's original algorithm converges q-superlinearly, and the intent is to conserve this property, the SDTR algorithm should omit Step 2 when q-superlinear convergence may be evident. According to Yuan (1995), the trust region algorithm will not converge at a q-superlinear rate until the trust region bound becomes unnecessary and successive steps become shorter,

$$\|\mathbf{s}^k\| < \Delta^k \quad \text{for all } k \text{ greater than some } k_0, \qquad (13)$$

$$\|\mathbf{s}^{k+1}\| < \|\mathbf{s}^k\| \quad \text{for all } k \text{ greater than some } k_1. \qquad (14)$$

If these conditions hold for a few consecutive steps, then Step 2 is omitted until convergence or until some iteration occurs where the conditions fail.

4.3 A correction step

Because the penalty function (3) is not smooth, SDTR may be subject to the trust region analogy of the Maratos effect described by Yuan (1984). In practice it is necessary to use a correction step, as suggested by Fletcher (1981). Further details can be found in the paper by Yuan (1995). Incorporating the correction step is straightforward but is omitted here for clarity. The correction step is used in the examples in the next section.

5 Examples

In this section, two moderately sized NLP are decomposed and solved with both the original and the modified version of the trust region algorithm. In each case, the NLP is solved using the same starting points, tolerances, and algorithm parameters. Some general comments are made after the examples are presented.

5.1 A propane combustion problem

The first example is taken from the NASA-Langley's MDO test suite (see Padula *et al.* 1996). Originally, the object is to find $\mathbf{x} \in \Re^{11}$ such that $f_i = 0$ for $i = 1, 2, \ldots, 11$. By defining the slack variables x_{12} and x_{13}, and two additional functions, f_{12} and f_{13}, the problem can be decomposed into two subproblems.

The problem can be stated as an unconstrained optimization problem by minimizing the sum of the squares of f_i for $i = 1, 2, \ldots, 13$. The equations are given below

$$f_1 = x_1 + x_4 - 3\,,$$

$$f_2 = 2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} - R\,,$$

$$f_3 = 2(x_2 + x_5) + x_6 + x_7 - 8\,, \quad f_4 = 2x_3 + x_9 - 4R\,,$$

$$f_5 = K_5 x_2 x_4 - x_1 x_5\,, \quad f_6 = K_6\sqrt{x_2 x_4} - x_6\sqrt{\frac{Px_2}{x_{11}}}\,,$$

$$f_7 = K_7\sqrt{x_1 x_2} - x_7\sqrt{\frac{Px_4}{x_{11}}}\,,$$

$$f_8 = K_8 x_1 - \frac{Px_4 x_8}{11} f_9 = K_9 x_1\sqrt{x_3} - x_4 x_9\sqrt{\frac{Px_4}{x_{11}}}\,,$$

$$f_{10} = K_{10} x_1^2 - \frac{Px_4^2 x_{10}}{11}\,,$$

$$f_{11} = x_{11} - (x_1 + x_2 + x_4 + x_7 + x_{12} + x_{13})\,,$$

$$f_{12} = x_{12} - (x_5 + x_6)\,, \quad f_{13} = x_{13} - (x_3 + x_8 + x_9 + x_{10})\,. \tag{15}$$

When decomposed, the first and second subproblems use the variables $\mathbf{x}_1 = [x_5, x_6]^T$ and $\mathbf{x}_2 = [x_3, x_8, x_9, x_{10}]^T$, respectively,

$$\min_{\mathbf{x}_1 \in \Re^2} \sum_{j=3,5,6,12} f_j^2(\mathbf{x}_0, \mathbf{x}_1)\,, \tag{16}$$

$$\min_{\mathbf{x}_2 \in \Re^4} \sum_{j=2,4,8,9,10,13} f_j^2(\mathbf{x}_0, \mathbf{x}_2)\,. \tag{17}$$

The linking variables are then $\mathbf{x}_0 = [x_1, x_2, x_4, x_7, x_{11}, x_{12}, x_{13}]^T$ and the remaining functions are grouped together as subproblem 0

$$\sum_{j=1,7,11} f_j^2(\mathbf{x}_0)\,. \tag{18}$$

In the computations, the values $\Delta = 1.0$, $\sigma = 10.0$, $\delta = 0.1$, $P = 40.0$, $R = 10.0$, $K_5 = 1.0$, $K_6 = 1.0$, $K_7 = 1.0$, $K_8 = 0.1$, $K_9 = 1.0$, and $K_{10} = 0.1$ are used, along with the starting point $\mathbf{x} = [1, 1, 15, 1, 1, 1, 1, 1, 1, 1, 24, 3, 15]^T$.

The value of the objective functions and the distance from the solution are shown in Figs. 1 and 2, respectively.

In Figs. 1 and 2, one increment along the abscissa represents a single function call or gradient call to any subproblem, so that the computational cost is measured fairly. The symbol $\times$ denotes a function call made during the execution of the first subproblem, the symbol $\circ$ denotes a function call during the execution of the second subproblem. All other function calls are made during the execution of the SDTR algorithm. Thus, one successful iteration of the TR algorithm will move six increments: once for evaluating the functions of subproblem 1, once for evaluating the gradients of subproblem 1, and so forth. From both figures, one can see that the modified algorithm uses fewer function and gradient calls to arrive at a solution.

5.2 A parking brake problem

The next example concerns the design of a parking brake and is adapted from the work of Krishnamachari (1996),

$$\min_{\mathbf{x} \in \Re^{14}} \; (100x_4 x_5 - x_9 - x_{11}) + (100x_6 x_7 - 30x_6 x_7^3)\,,$$

$$g_1(\mathbf{x}) = 2.94 - 2K_7 x_7 x_8 x_{10} x_{13} \leq 0\,,$$

$$g_2(\mathbf{x}) = K_8 - 2K_7 x_7 x_8 x_{10} x_{13} \leq 0\,,$$

$$g_3(\mathbf{x}) = x_1 - x_3 + 0.45x_8 \leq 0\,,$$

$$g_4(\mathbf{x}) = \frac{x_5 x_{12}}{2} - K_{10} x_{11} \leq 0\,,$$

$$g_5(\mathbf{x}) = K_{11} x_2 - x_1 x_6 x_7 \leq 0\,,$$

$$g_6(\mathbf{x}) =$$

$$\frac{K_{12} x_2 x_9}{x_1} - \frac{x_6 x_7^3}{\left[0.45x_8\left(1.25 + \frac{x_1}{x_3 - 0.45x_8}\right)\right]^2} \leq 0\,,$$

subject to

$$g_7(\mathbf{x}) = x_1 - x_2 + 5 \leq 0\,,$$

$$h_1(\mathbf{x}) = x_9 - 2K_1 x_8 x_{10} x_{13} = 0\,,$$

$$h_2(\mathbf{x}) = x_1 x_8 x_{10} + \frac{5}{9}(x_2(x_3 - x_1) - x_3(x_2 - x_1)) = 0\,,$$

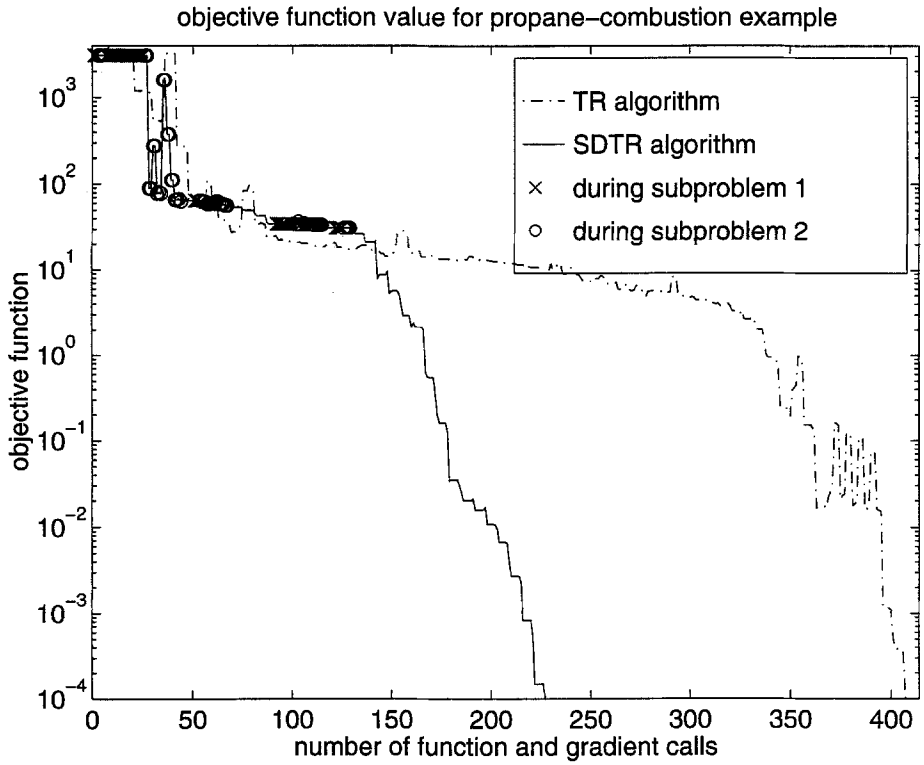

Fig. 1. Value of the objective function of the propane combustion problem as the iterates progress. One increment along the abscissa denotes either a function call or a gradient call to any one of the subproblems

$$h_3(\mathbf{x}) = x_{11} - \frac{x_4 x_5^3}{12} = 0\,,$$

$$h_4(\mathbf{x}) = x_{12} - K_4(x_2 - x_1) = 0\,,$$

$$h_5(\mathbf{x}) = x_{13} - (x_8 x_{14}) = 0\,,$$

$$h_6(\mathbf{x}) = x_{14} - 0.8\frac{x_3}{x_8} = 0\,. \tag{19}$$

The first subproblem uses $\mathbf{x}_1 = [x_4, x_5, x_{11}, x_{13}]^T$

$$\min_{\mathbf{x}_1 \in \Re^4} \quad f_1 = 100 x_4 x_5 - x_9 - x_{11}\,,$$

subject to

$$g_1(\mathbf{x}_0, \mathbf{x}_1) \leq 0 \; h_1(\mathbf{x}_0, \mathbf{x}_1) = 0\,,$$

$$g_3(\mathbf{x}_0, \mathbf{x}_1) \leq 0 \; h_2(\mathbf{x}_0, \mathbf{x}_1) = 0\,,$$

$$g_5(\mathbf{x}_0, \mathbf{x}_1) \leq 0 \; h_4(\mathbf{x}_0, \mathbf{x}_1) = 0\,,$$

$$5 \leq x_4 \leq 20\,, \quad 5 \leq x_5 \leq 30\,. \tag{20}$$

The second subproblem uses $\mathbf{x}_2 = [x_1, x_2, x_3, x_6, x_7]^T$

$$\min_{\mathbf{x}_2 \in \Re^6} \quad f_2 = 100 x_6 x_7 - 30 x_6 x_7^3\,,$$

subject to

$$g_2(\mathbf{x}_0, \mathbf{x}_2) \leq 0 \; h_3(\mathbf{x}_0, \mathbf{x}_2) = 0\,,$$

$$g_4(\mathbf{x}_0, \mathbf{x}_2) \leq 0 \; h_5(\mathbf{x}_0, \mathbf{x}_2) = 0\,,$$

$$g_6(\mathbf{x}_0, \mathbf{x}_2) \leq 0 \; h_6(\mathbf{x}_0, \mathbf{x}_2) = 0 \; h_7(\mathbf{x}_0, \mathbf{x}_2) = 0\,,$$

$$10 \leq x_1 \leq 24\,, \quad 10 \leq x_2 \leq 102\,, \quad 3 \leq x_3 \leq 183\,,$$

$$5 \leq x_6 \leq 20\,, \quad 5 \leq x_7 \leq 30\,. \tag{21}$$

Although the linking variables are $\mathbf{x}_0 = [x_8, x_9, x_{10}, x_{12}, x_{14}]^T$, the only remaining functions are the simple bounds on x_8,

$$160 \leq x_8 \leq 240\,. \tag{22}$$

The example was performed using the values $\Delta = 10.0$, $\sigma = 10.0$, and $\delta = 0.1$, the parameters $K_1 = 0.8748$, $K_4 = 648.0$, $K_7 = 0.000459$, $K_8 = 1221.94$, $K_{10} = 324.0$, $K_{11} = 2.0$, $K_{12} = 0.000211$, and the starting point $\mathbf{x} = [15,10,180,15,\ 15,15,10,170,14580,0.16340,4218.8,\ -3240,144,0.8470]^T$. The value of the objective function, and the distance from the solution are shown in Figs. 3 and 4, respectively. As the figures show, the modified algorithm uses fewer function and gradient calls to arrive at a solution.

5.3 Discussion of examples

In both examples, we see that the SDTR algorithm reaches a solution using significantly fewer function calls than the

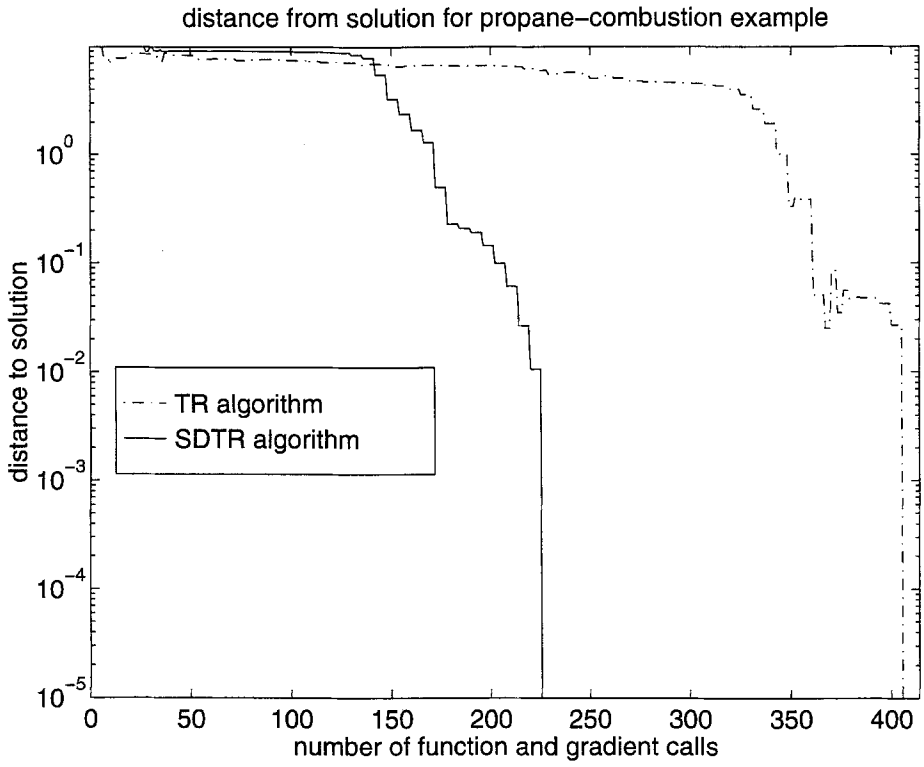

Fig. 2. Distance from the solution of the propane combustion problem as the iterates progress. One increment along the abscissa denotes either a function call or a gradient call to any one of the subproblems.

Table 1. Number and types of function calls and other calculations made for each of the examples

	Propane		Parking Brake	
	TR	SDTR	TR	SDTR
Function calls to subproblem 1	138	82	164	65
Function calls to subproblem 2	138	103	164	54
Function calls to subproblem 0	138	49	164	37
Gradient calls to subproblem 1	48	36	60	27
Gradient calls to subproblem 2	48	42	60	33
Gradient calls to subproblem 0	48	24	60	13
Number of solutions required for (6)	138	49	164	37

original TR algorithm. If these results hold for larger and more computationally expensive models, then SDTR would be worthwhile even without the explicit use of parallelism. To quantify the computational savings, the number and type of function calls made during the execution of each algorithm is given in Table 1.

The most notable difference is the reduction in the number of function and gradient calls made to subproblem 0. This is not surprising because analyses to subproblem 0 are only made during the outer loop (Step 4 in Algorithm 4). The number of function and gradient calls made to subproblems 1 and 2 has also decreased, even though analysis may be performed when solving the subproblems (Step 2) or by the outer loop (Step 4). For very large NLP, the bottleneck occurs in solving the large approximate problem [in SQP the QP becomes problematic, and in Yuan's algorithm solving (6) becomes problematic]. The SDTR algorithm still needs a solution to (6), but it is required fewer times. [The subproblems also need solutions to (6), but the size of the problem is much smaller.]

With these potential benefits, one may wonder about the specific causes for the improvements. The convergence analysis (see Nelson 1997) does not prove any benefits for the SDTR algorithm, only that SDTR has the same convergence

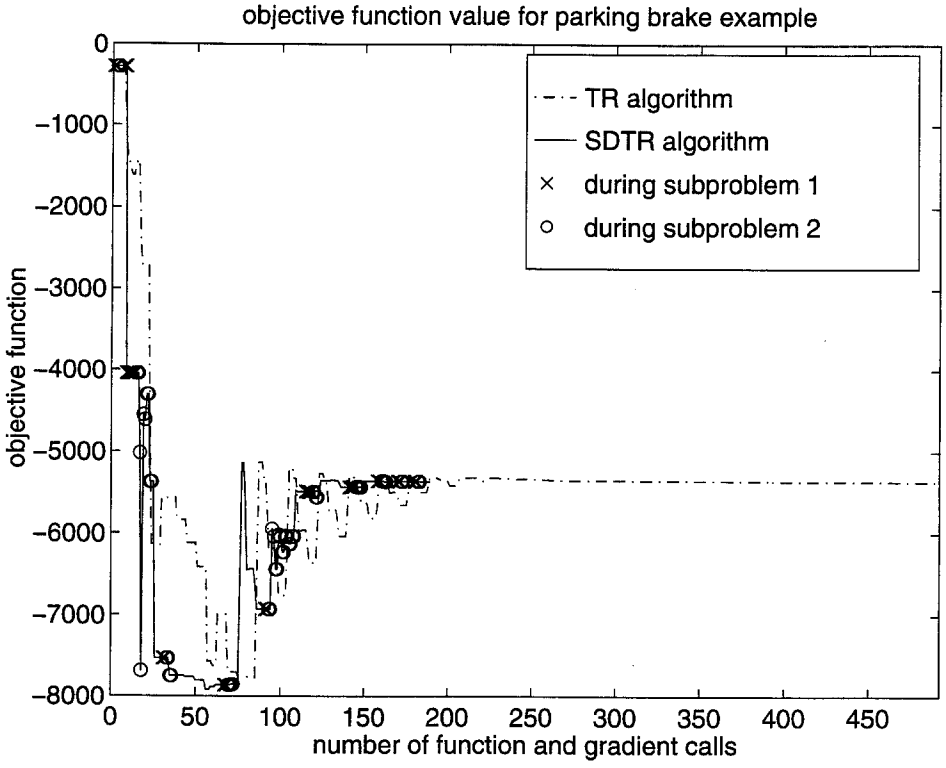

Fig. 3. The value of the objective function for the parking brake problem as the iterates progress. One increment along the abscissa denotes either a function call or a gradient call to any one of the subproblems

properties as the TR algorithm. However, there are some plausible hypotheses for the benefits shown in the examples.

First, *valid progress is made during the execution of a subproblem.* This can be seen, for example, by examining Fig. 1. Between the twentieth and fiftieth function call, subproblem 2 makes a significant reduction in the objective function. It is interesting to note that in Fig. 2 the iterates have not necessarily moved closer to the solution $\mathbf{x}^*$, and we remind the reader that ensuring global convergence does not require improving $\|\mathbf{x}-\mathbf{x}^*\|$ (see Nelson and Papalambros 1997). Additionally, if one subproblem requires more work to find a solution than the other subproblems, SDTR can focus the effort on the subproblems in need, whereas TR must focus on the problem as a whole.

Second, *the subproblems improve estimates of parameters used during coordination.* Admittedly, this is difficult to prove but some indirect arguments are possible.

The first argument concerns the Hessian estimate: a Hessian estimate effectively alters the search direction from an inefficient steepest descent direction (for an inaccurate Hessian and a small trust region) to the very efficient Newton direction (for an exact Hessian). The way that SDTR is constructed, the Hessian may be updated by the subproblems several times prior to the first coordination problem. When coordination finally occurs, the Hessian estimate is much more accurate than the identity matrix, which is usually the initial Hessian estimate for the unmodified algorithm.

The second argument, although not as concrete, involves the interplay between the trust region radius and the accuracy of the penalty function. Intuitively, if the size of the trust region was appropriately set by a subproblem, unnecessary backtracking could be avoided during coordination.

6 Conclusions

The SDTR algorithm offers a simple way of exploiting a hierarchically decomposed structure. As the theoretical properties are reviewed elsewhere, this paper focused on a description of the algorithm and the performance during computation. The SDTR algorithm is constructed so that the convergence properties of the TR algorithm are conserved, but this does not prove that SDTR is more efficient than TR.

However, the numerical examples do indicate that the SDTR algorithm is an improvement in that fewer function and gradient calls are needed to arrive at a solution when compared to the TR algorithm. Several possible hypotheses for the improvement were stated, including the facts that the SDTR algorithm uses subproblems to improve algorithmic parameters and that SDTR has the ability to concentrate on a particular subproblem when necessary. It is believed that these results will extend to larger and more difficult NLP.

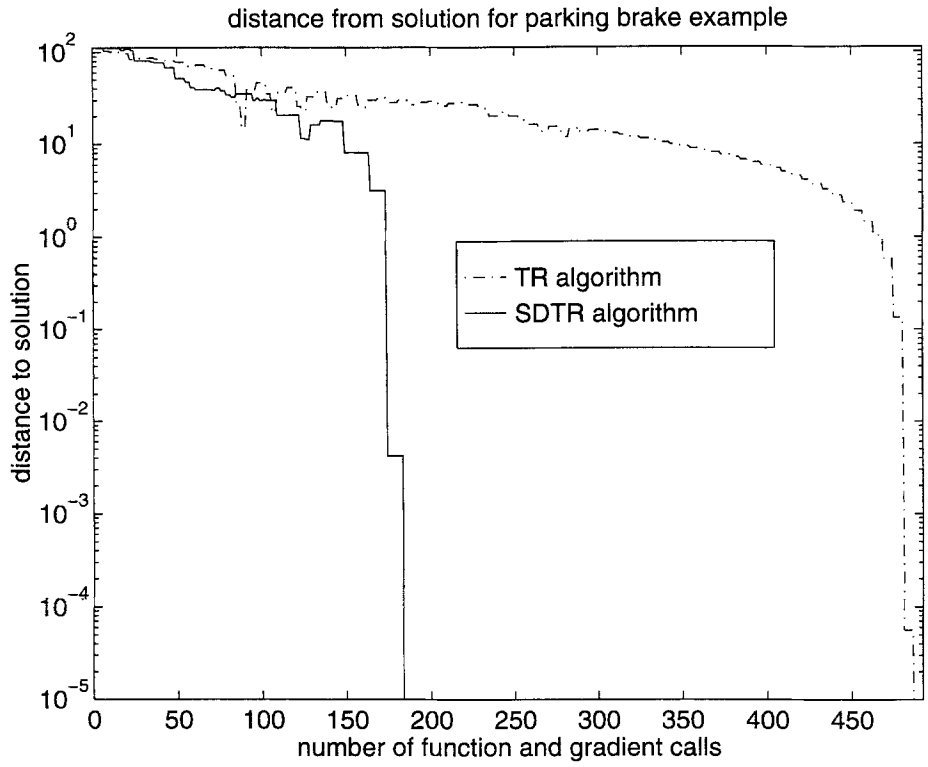

Fig. 4. Distance from the solution of the parking brake problem as the iterates progress. One increment along the abscissa denotes either a function call or a gradient call to any one of the subproblems

Acknowledgements

This research has been partially supported by the Automotive Research Center at the University of Michigan, a US Army Center of Excellence in Modeling and Simulation of Ground Vehicles, under Contract No. DAAE07-94-C-R094. This support is gratefully acknowledged. The authors would also like to thank R. Krishnamachari and H.M. Kim for developing the parking brake example.

References

Conn, A.R.; Gould, N.I.M.; Toint, P.L. 1992: *LANCELOT, A FORTRAN package for large-scale nonlinear optimization.* Berlin, Heidleberg, New York: Springer

Fletcher, R. 1981: Second order corrections for nondifferentiable optimization. In: Watson, G.A. (ed.) *Numerical analysis, Dundee 1981*, pp. 85–115. Berlin, Heidelberg, New York: Springer

Griewank, A. 1991: The global convergence of partitioned BFGS on problems with convex decompositions and Lipschitzian gradients. *Math. Prog.* **50**, 141–175

Griewank, A.; Toint, P.L. 1982a: Local convergence analysis for partitioned quasi-Newton updates. *Numerische Mathematik* **39**, 429–448

Griewank, A.; Toint, P.L. 1982b: Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik* **39**, 119–137

Krishnamachari, R. 1996: *A decomposition synthesis methodology for optimal system design.* Ph.D. Thesis, University of Michigan, Ann Arbor

Krishnamachari, R.; Papalambros, P.Y. 1997: A decomposition synthesis methodology for optimal system design. *ASME J. Mech. Des.* (to appear)

Michelena, N.; Papalambros, P.Y. 1995: Optimal model-based decomposition of powertrain system design. *ASME J. Mech. Des.* **117**, 499–505

Nelson, S.A., II 1997: *Optimal hierarchical system design via sequentially decomposed programming.* Ph.D. Thesis, The University of Michigan, Ann Arbor

Nelson, S.A., II; Papalambros, P.Y. 1997: Sequentially decomposed programming. *AIAA J.* **35**, 1209–1216

Padula, S.L.; Alexandrov, N.; Green, L.L. 1996: MDO test suite at NASA Langley Research Center. In: *Proc. 6-th AIAA/NASA/ISSMO Symp. on Multipdisciplinary Analysis and Optimization* (held in Bellevue, WA), pp. 410–420. AIAA

Papalambros, P.Y. 1995: Optimal design of mechanical engineering systems. *ASME J. Mech. Des.* **117**, 55–62

Powell, M.J.D. 1975: Convergence properties of a class of minimization algorithms. In: Mangasarian, O.L.; Meyer, R.R.; Robinson, S.M. (eds.) *Nonlinear programming 2*, pp. 1–27. New York: Academic Press

Sobieszczanski-Sobieski, J.; James, B.B.; Riley, M.F. 1987: Structural sizing by generalized multilevel optimization. *AIAA J.* **25**, 139–145

Thareja, R.R.; Haftka, R.T. 1990: Efficient single-level solution of hierarchical problems in structural optimization. *AIAA J.* **28**, 506–514

Yuan, Y.-X. 1984: An example of only linearly convergence of trust region algorithms for nonsmooth optimization. *IMA J. Numer. Anal.* **4**, 327–335

Yuan, Y.-X. 1995: On the convergence of a new trust region algorithm. *Numerische Mathematik* **70**, 515–539

Received Jan. 1, 1998
Communicated by J. Sobieski