Computer Assisted Mechanics and Engineering Sciences, 6: 469–477, 1999.

Nonlinear constrained optimizer and parallel processing for golden block line search

Duc T. Nguyen[1], Wilson H. Tang[2], Yeou K. Tung[2]
and Hakizumwami B. Runesha[1]

[1] *Multidisciplinary Parallel-Vector Computation Center, Kaufman 135, Old Dominion University, Norfolk, VA 23529 (USA)*

[2] *Department of Civil and Structural Engineering Hong Kong University of Science and Technology, Kowloon, Hong Kong*

(Received March 8, 1999)

Generalized exponential penalty functions are constructed for the multiplier methods in solving nonlinear programming problems. The non-smooth extreme constraint Gext is replaced by a single smooth constraint Gs by using the generalized exponential function (base a > 1). The well-known K.S. function is found to be a special case of our proposed formulation. Parallel processing for Golden block line search algorithm is then summarized, which can also be integrated into our formulation. Both small and large-scale nonlinear programming problems (up to 2000 variables and 2000 nonlinear constraints) have been solved to validate the proposed algorithms.

1. INTRODUCTION

Consider the following nonlinear programming (NLP) problem:

$$\text{(A)} \qquad \min f(x)$$
$$\text{s.t. } g_i(x) \leq 0, \qquad (i = 1, 2, \ldots, m),$$

or

$$\text{(A1)} \qquad \min f(x)$$
$$\text{s.t. } g_i(x) \geq 0, \qquad (i = 1, 2, \ldots, m),$$

where $f(x)$ and $g_i(x)$ are smooth, nonlinear functions of x ($x \in R^n$).

Although many methods have been developed for solving the above constrained nonlinear programming problems, such as the barrier and penalty function sequential methods [2] and the multiplier method [3, 4], they either have computational difficulties or have to deal with a non-smooth combined objective function. Recently, the exponential function was used to provide a smooth cumulative constraint function in the combined objective function for the multiplier method [5, 6]. This exponential function, known as $KS(x)$ function [1] has already been used in optimal structural synthesis and designs [7, 8, 9, 10]. One disadvantage of using the $KS(x)$ function, however, is that the user has to choose a suitable parameter p. If p is too small then the results will not be accurate, if p is too large then the $KS(x)$ function will become non-smooth [11]. However, p **should be gradually increased to a large positive value** in its derivation given in [5, 6]. In this paper, the first (major) objective is to construct a generalized exponential function g_s such that p is no longer needed as an accuracy controlling parameter, and the $KS(x)$ function is included as its special form. This proposed generalized exponential function g_s is used to obtain a single constraint in the multiplier method to provide a smooth composite objective function. The second

(minor) objective of this paper is to develop and to validate the parallel processing procedure for golden block line search which can be incorporated into the first objective to enhance the numerical performance. Several nonlinear programming and line search problems [12] are solved by using this generalized exponential function. The largest problem solved involves 2000 variables and 2000 nonlinear constraints.

2. Derivations of the generalized exponential function

Problem (A) is equivalent to the following nonlinear programming problem:

$$\text{(B)} \qquad \min f(x)$$
$$\text{s.t. } g_{\max}(x) \leq 0$$

Similarly, problem (A1) is equivalent to

$$\text{(B1)} \qquad \min f(x)$$
$$\text{s.t. } g_{\min}(x) \geq 0$$

where the single constraint $g_{\max}(x)$ (or $g_{\min}(x)$) is defined as the "extreme" constraint which can be expressed as

$$g_{\text{ext}}(x) = g_{\max}(x) = \max\{g_i(x)\} \qquad \text{(for problem (B)}, \quad i = 1, 2, \ldots, m)$$
$$= g_{\min}(x) = \min\{g_i(x)\} \qquad \text{(for problem (B1)}, \quad i = 1, 2, \ldots, m).$$

When the original problem (A) or (A1) has at least one active constraint, then the problem (B) or (B1) can be solved by means of multiplier penalty method [4]:

$$\text{(C)} \qquad \text{minimize } \phi(x, \alpha) = f(x) + \alpha g_{\text{ext}}(x) + \frac{c}{2} g_{\text{ext}}^2(x)$$

where α is a Lagrange multiplier and c is a penalty factor, α should be updated during the iterations by the formula:

$$\alpha_{k+1} = \alpha_k + c\, g_{\text{ext}}(x_k)$$

where x_k is the solution of (C) at step k, and c can be kept constant or can be increased gradually. It should also be noted here that x_k in the above formula should contain the updated values upon exiting from a successful unconstrained minimization.

However, the use of g_{ext} can cause difficulties in the solution of (C), since g_{ext} is not a smooth function. We will try to construct a smooth function $g_s(x)$ to replace $g_{\text{ext}}(x)$ so that to avoid those difficulties. It is obvious that $g_s(x)$ should have the following properties:

1. $g_s(x) = g_{\text{ext}}(x)$ (if $m = 1$)
   $g_s(x) = g_{\text{ext}}(x) + \varepsilon(x)$ (if $m > 1$)
2. $g_s(x) = G(g_i(x)), \quad i = 1, \ldots, m.$

The first property of $g_s(x)$ assures $g_s(x)$ is a good approximation of $g_{\text{ext}}(x)$, while the second property suggests $g_s(x)$ is a function of $g_i(x)$ $(i = 1, 2, \ldots, m)$. In other words, $g_i(x)$ are the basic variables of $g_s(x)$. The smooth function $g_s(x)$ can be determined by first define a generalized exponential function $F = F(g_i(x), a)$, such as

$$F = \sum_{I=1}^{m} a^{p g_i(x)} \qquad (a > 1, \ p \neq 0).$$

It can be seen that $a^{pg_i(x)}$ ($a > 1$) is a monotone increasing function of $g_i(x)$ when $p > 0$, and a monotone decreasing function of $g_i(x)$ when $p < 0$. Thus, if a is large enough then F is a good approximation of $a^{pg_{\text{ext}}}$. But we only need the approximation of $g_{\text{ext}}(x)$, so we have to invert F to get $g_{\text{ext}}(x)$. Since $F = a^{pg_{\text{ext}}}$, this suggests that $g_s(x)$ should take the form

$$g_s(x) = g_s(g_i(x)) = \frac{1}{p}\log_a F = \frac{1}{p}\log_a\left(\sum_{i=1}^{m} a^{pg_i(x)}\right) = g_{\text{ext}}(x) + \frac{1}{p}\log_a\left(\sum_{i=1}^{m} a^{p(g_i(x)-g_{\text{ext}}(x))}\right).$$

It is found that $g_s(x)$ has the following property:

$$g_{\max}(x) \le g_s(x) \le g_{\max}(x) + \frac{1}{p}\log_a m \qquad (p > 0, \quad g_{\text{ext}} = g_{\max}),$$

$$g_{\min}(x) + \frac{1}{p}\log_a m \le g_s(x) \le g_{\min}(x) \qquad (p < 0, \quad g_{\text{ext}} = g_{\min}).$$

If we require the error term $(1/|p|)\log_a m$ less than ε , then the base a can be determined by

$$a > m^{\frac{1}{|p|\varepsilon}}.$$

The $KS(x)$ function as it is used in [1, 5, 6, 7, 8, 9, 10] is defined as

$$KS(x) = \frac{1}{p}\ln\left(\sum_{i=1}^{m} e^{pg_i(x)}\right).$$

It is interesting to note that the $KS(x)$ function is a special form of the proposed $g_s(x)$ function. One only has to set the special base: $a = e$. However, the advantage of using the $g_s(x)$ function is that we do not have to adjust the controlling parameter p as it is required in the $KS(x)$ function. Furthermore, we can control the smoothness of the $g_s(x)$ function by using different bases ($a > 1$) during the iterations. In fact, the non-zero parameter p is only used as a sign for the extreme constraint ($g_{\text{ext}} = g_{\max}$, if $p > 0$; or $g_{\text{ext}} = g_{\min}$, if $p < 0$). However, p must be gradually increased to a large positive value in the derivation given in [5, 6], and the $KS(x)$ function will fail if $p \ll 1$. Since higher accuracy in the constraint conditions is desired in later iterations, it is suggested that the base a should be increased during the iterations.

3. A STEP-BY-STEP NONLINEAR PROGRAMMING ALGORITHM

When the extreme constraint in problem (B) or (B1) is replaced by a single constraint $g_s(x)$ as defined above, we now can solve the following constrained nonlinear programming problem (assume problem (B) is now under consideration)

(D) $\min f(x)$

s.t. $g_s(x) \le 0$

through the solution of the following sequential unconstrained nonlinear programming problem:

(E) minimize $\phi(x, \alpha) = f(x) + \alpha g_s(x) + \frac{c}{2} g_s^2(x)$

where any standard unconstrained NLP methods (such as BFGS, or DFP) [13, 14] can be used (BFGS method is used in this paper, see the Appendix).

A simple NLP algorithm can be written as:

1. Set initial values for base a and c, and x_0 , let $\alpha_0 = 0$, $k = 0$.
2. Minimize $\phi(x, c)$ using any standard unconstrained NLP methods.

3. Convergence check: $\|x_{k+1} - x_k\| < c$, if satisfied go to 5.
4. Update α, a and c by:

$$\alpha_{k+1} = \alpha_k + c_k g_s(x_k) \qquad (x_k \text{ is the minimizer}),$$
$$a_{k+1} = 10^{1000} a_k\,,$$
$$c_{k+1} = \max(1000000, 1.1c_k),$$

go to 2.

5. stop.

4. PARALLEL GOLDEN BLOCK SEARCH TECHNIQUE

In this section, a parallel version of the golden block search technique has been developed for determining the step size α (see Step 4 of the Appendix). Theoretical development of the golden block search technique can be summarized in the following paragraphs:

- The golden search method is based on the Fibonacci sequence, which is defined as

$$F_0 = 1; \qquad F_1 = 1; \qquad F_n = F_{n-1} + F_{n-2} \qquad (n = 2, 3, 4, \ldots),$$

with the properties

$$\left.\frac{F_n}{F_{n-1}}\right|_{n\to\infty} = \tau = \frac{1}{2}(1 + \sqrt{5}) \approx 1.618 = \text{golden ratio}.$$

- The Fibonacci sequence is a special case of the Arriel sequence

$$A_k^0 = 1; \qquad A_k^1 = 1; \qquad A_k^n = k(A_k^{n-1} + A_k^{n-2}) \qquad (n = 2, 3, 4, \ldots). \tag{1}$$

Thus, when $k = 1$, then the Arriel sequence will become the Fibonacci sequence.

- In order to apply the Arriel sequence to modify the golden search technique, we assume:

$$\frac{A_k^{n+2}}{A_k^{n+1}} = \frac{A_k^{n+1}}{A_k^n} = \tau_k \quad \text{as} \quad n \to \infty \tag{2}$$

and try to derive the condition for which τ_k (refer to Eq. (2)) needs to be satisfied.

- Derivation of a formula for τ_k. Multiplying $\frac{A_k^{n+1}}{A_k^n}$ to both sides of Eq. (2)

$$\frac{A_k^{n+2}}{A_k^{n+1}} \frac{A_k^{n+1}}{A_k^n} = \left(\frac{A_k^{n+1}}{A_k^n}\right)^2 = \tau_k^2\,. \tag{3}$$

From Eq. (1), one has:

$$A_k^{n+2} = k(A_k^{n+1} + A_k^n). \tag{4}$$

Substituting Eq. (4) into 3, one obtains:

$$\frac{k(A_k^{n+1} + A_k^n)}{A_k^n} = \tau_k^2 = k\left(\frac{A_k^{n+1}}{A_k^n} + 1\right)$$

or

$$\tau_k^2 = k(\tau_k + 1). \tag{5}$$

Solving the quadratic Eq. (5) and using only the positive root, one has:

$$\tau_k = \frac{1}{2}\left(k + \sqrt{k^2 + 4k}\right). \tag{6}$$

Note: if $k = 1$, then $\tau_k = \frac{1}{2}(1 + \sqrt{5}) = 1.618 =$ the standard golden search ratio.

The above golden block search algorithm can be conveniently presented in a form of a step-by-step algorithm.

Step 1: $d_B^0 = b - a$, where a and b are estimated lower and upper bounds for the step size α.

Step 2: First block search (for $i = 1$)

- $\alpha_0^1 = a$,
- $\alpha_1^1 = a + \dfrac{1}{\tau_k} d_B^0$ where $\tau_k = \frac{1}{2}(k + \sqrt{k^2 + 4k})$,
- $\alpha_j^1 = \alpha_{j-2}^1 + \dfrac{1}{k} d_B^0$ where $j = 2, 3, \ldots, 2k$,
- Parallel computation for $F(\alpha_j^1)$ where $j = 0, 1, 2, 3, \ldots, 2k$.

Step 3: Find the value of α_j^1 which gives the minimum value of F, say $\alpha_j^1 = \alpha_l^1$.

Step 4: Set $r_1 = \alpha_{l-1}^1$ and $R_1 = \alpha_{l+1}^1$. Thus $d_B^1 = R_1 - r_1 = \frac{d_B^0}{k}$.

Step 5: Subsequent i^{th} block search (for $i \geq 2$)

- $\alpha_0^i = r_{i-1}$,
- $\alpha_1^i = r_{i-1} + \left(\dfrac{1}{\tau_k}\right)^i d_B^0$,
- $\alpha_j^i = \alpha_{j-2}^i + \dfrac{d_B^0}{k} \tau_k^{1-i}$ where $j = 2, 3, \ldots, 2k$,
- compute $F(\alpha_j^i)$.

Step 6: Return to step 3 if the process does not converge.

Based upon the above step-by-step procedure, the parallel golden block search algorithm has been developed, and validated as shown in Section 5.

5. Numerical applications

Several examples are solved using the above algorithms, the first two examples are taken from [12], while the last two examples are constructed by the authors to test the capability of the present algorithms. For all these examples, $p = 1$ and the initial values for a and c are $a_0 = 10^{100}$, $c_0 = 10000$, and $\varepsilon = 0.0001$. The equality constraint $g(x) = 0$ is replaced by $g(x) \leq 0$ and $-g(x) \leq 0$.

Example 1: Problem number 66 (see [12], p. 88).

$$\begin{aligned}
\min\ & f(x) = 0.2x_3 - 0.8x_1 \\
\text{s.t.}\ & e^{x_1} - x_2 \le 0, \\
& e^{x_2} - x_3 \le 0, \\
& 0 \le x_1 \le 100, \\
& 0 \le x_2 \le 100, \\
& 0 \le x_3 \le 10.
\end{aligned}$$

Optimum solution as well as iterations history are presented in Table 1.

Table 1. Computational results for Example 1

Iteration No.	$\|x_{k+1} - x_k\|$	$\|\Phi_k\|$	$g_{\max}$	$f(x_k)$
0	—	70927.3	1.0	−0.00006
1	3.59349	$6.89 \cdot 10^{-2}$	$-1.0457 \cdot 10^{-3}$	0.52028132
2	$5.2368 \cdot 10^{-2}$	$3.092 \cdot 10^{-3}$	$-1.0357 \cdot 10^{-4}$	0.51834827
3	$1.9833 \cdot 10^{-4}$	$1.708 \cdot 10^{-2}$	$-5.3822 \cdot 10^{-5}$	0.51825999
4	$1.5833 \cdot 10^{-4}$	$1.019 \cdot 10^{-2}$	$-3.6524 \cdot 10^{-5}$	0.51822897
5	$8.6909 \cdot 10^{-5}$	$2.830 \cdot 10^{-2}$	$-2.7202 \cdot 10^{-5}$	0.51821282

*$x^0 = \{0.0001, \ldots, 0.0001\}$ (not feasible);
Ref. [12]'s optimum: $f = 0.518163274$.

Example 2: Problem number 100 (see [12], p.111).

$$\begin{aligned}
\min\ & f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\
& \quad + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\
\text{s.t.}\ & g_1(x) = 2x_1^2 + 3x_2^4 + x_3 + 4x_4^3 + 5x_5 - 127 \le 0, \\
& g_2(x) = 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 \le 0, \\
& g_3(x) = 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 \le 0, \\
& g_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \le 0.
\end{aligned}$$

Again, the solutions during the iterations are given in Table 2.

Table 2. Computational results for Example 2

Iteration No.	$\|x_{k+1} - x_k\|$	$\|\Phi_k\|$	$g_{\max}$	$f(x_k)$
0	—	741.26	0.0006	1183.0224
1	5.952827	7287.57	$2.7630 \cdot 10^{-3}$	741.596
2	1.264876	10506.29	$-4.4204 \cdot 10^{-4}$	700.750
3	0.683534	5618.36	$-1.2526 \cdot 10^{-3}$	686.735
4	0.767874	2321.53	$-2.1536 \cdot 10^{-4}$	682.340
5	0.632013	170.833	$-3.0169 \cdot 10^{-4}$	680.663
6	$7.137 \cdot 10^{-2}$	128.389	$-6.4009 \cdot 10^{-5}$	680.636
7	$4.382 \cdot 10^{-3}$	165.360	$2.9859 \cdot 10^{-5}$	680.635
8	$1.224 \cdot 10^{-2}$	310.711	$-4.4446 \cdot 10^{-5}$	680.633
9	$4.6328 \cdot 10^{-6}$	21.744	$-4.642 \cdot 10^{-5}$	680.633

*$x^0 = \{-0.0001, \ldots, -0.0001\}$ (not feasible); Ref. [12]'s optimum: $f = 680.6300573$.

Example 3: Large-scale nonlinear programming test problem.

$$\min \quad f(x) = -\sum_{i=1}^{m} x_i^3$$

$$\text{s.t.} \quad g_i(x) = \sum_{j=1}^{m} x_j^2 + n x_i^2 - (2n-1) \leq 0, \qquad (i = 1, 2, \ldots, m; \quad j \neq i)$$

In this example, $n = m = 2000$, the optimum solution is: $x = \{1, \ldots, 1\}$ and the optimum value is $f = -n = -2000$. Numerical results are presented in Table 3. It should be noted here that an effective equation solver [15] (using Gaussian elimination) is used in the BFGS method (see step 2 of Section 3) in order to reduce the total solution time for this large-scale NLP problem.

Table 3. Computational results for Example 3

Iteration No.	$\|x_{k+1} - x_k\|$	$\|\Phi_k\|$	$g_{\max}$	$f(x_k)$
0	—	$9.345 \text{x} 10^{12}$	99	−2000000
1	402.5014	2926.69	$-1.7541 \cdot 10^{-5}$	−1998.77
2	$9.1612 \cdot 10^{-3}$	26093.2	$5.935 \cdot 10^{-4}$	−1999.997

*$x^0 = \{10, \ldots, 10\}$ (not feasible); optimum: $f = -2000$.

This problem is solved on Cray Y-MP (using one processor) with CPU time = 564.681 seconds.

Example 4: Parallel golden block search method.

Find t which minimizes the function

$$F(t) = \cos(t) = 1 - \frac{t^2}{2!} + \frac{t^4}{4!} - \frac{t^6}{6!} + \cdots + \frac{t^n}{n!} + \cdots$$

The optimum solution is $t = t^* = \pi$ and $F = F^* = -1.0$.

The performance of the parallel golden block search algorithm is shown in Table 4.

Table 4. Parallel golden block search example *(continued in the next page)*

$n = 600, \ \epsilon = 1.0 \cdot 10^{-9}, \ k = 1$

NP	Time (seconds)	S	η (%)
1	0.3553	1.00	100.0

$k = 2$

1	0.3381	1.00	100.0

$k = 3$

1	0.3866	1.00	100.0
2	0.2008	1.925	96.25
3	0.13668	2.83	94.30
4	0.12797	3.02	75.60

$k = 5$

1	0.48918	1.00	100.0
2	0.25147	1.95	97.30
3	0.19397	2.52	84.10
4	0.14565	3.36	84.00

NP – number of processors used

k – the coefficient given in Eq. (6)

n – number of terms used in the above function $F(t)$

S – speed up factor

η – efficiency

ϵ – convergence tolerance

Table 4. *(continued)* Parallel golden block search example

$k = 6$

NP	Time (seconds)	S	η (%)
1	0.54002	1.00	100.
2	0.27735	1.95	97.4
3	0.18711	2.89	96.2
4	0.14365	3.76	94.0

$k = 7$

1	0.57734	1.000	100.
2	0.29258	1.973	98.7
3	0.20595	2.8033	93.4
4	0.16478	3.504	87.6

6. Discussions and conclusions

A generalized exponential penalty function $g_s(x)$ is constructed where p is no longer needed as an accuracy controlling parameter as it is in the $KS(x)$ function. Rather, p is only introduced here to facilitate the treatment of the extreme constraint g_{ext} (either maximum or minimum). Furthermore, the $KS(x)$ function is considered as the special case of the proposed $g_s(x)$ function. Since the non-smooth extreme constraint $g_{\text{ext}}(x)$ is replaced by a smooth constraint $g_s(x)$, it is expected that the present algorithm will lead to better performance during the optimization process. In fact, in all the examples considered in this paper, optimum solutions are essentially obtained with less than 5 iterations. The proposed $g_s(x)$ function and parallel golden block line search can be used in a wide range of optimization problems.

Acknowledgement

The authors would like to acknowledge the helpful detailed comments from the reviewers who have pointed out several typing errors occurred in the formulas. The authors would also like to acknowledge the support of the Research Grant Council of Hong Kong through grant No. HKUST 6039/97E.

Appendix. BFGS algorithm

Step 1: Estimate an initial design $x^{(0)}$. Choose a symmetric positive definite matrix $H^{(0)}$ as an estimate for the Hessian of the cost function. In the absence of more information, let $H^{(0)} = I$. Choose a convergence parameter ϵ. Set $k = 0$, and compute the gradient vector as $c^{(0)} = \nabla f(x^{(0)})$ where f is an objective function.

Step 2: Calculate the norm of the gradient vector as $\|c^{(k)}\|$. If $\|c^{(k)}\| < \epsilon$ then stop the iterative process; otherwise continue.

Step 3: Solve the following linear system of equations to obtain the search direction:

$$H^{(k)} p^{(k)} = -c^{(k)}.$$

Step 4: Compute optimum step size $\alpha_k = \alpha$ to minimize $f(x^{(k)} + \alpha p^{(k)})$.

Step 5: Update the design as

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}.$$

Step 6: Update the Hessian approximation for the cost function as

$$H^{(k+1)} = H^{(k)} + D^{(k)} + E^{(k)}$$

where the correction matrices $D^{(k)}$ and $E^{(k)}$ are given as

$$D^{(k)} = \frac{y^{(k)} y^{(k)^T}}{y^{(k)} \bullet s^{(k)}}; \qquad E^{(k)} = \frac{c^{(k)} c^{(k)^T}}{c^{(k)} \bullet p^{(k)}}$$

with

$$\begin{aligned} s^{(k)} &= \alpha_k p^{(k)} && \text{(change in design)}, \\ y^{(k)} &= c^{(k+1)} - c^{(k)} && \text{(change in gradient)}, \\ c^{(k+1)} &= \nabla f(x^{(k+1)}). \end{aligned}$$

Step 7: Set $k = k + 1$ and go to step 2.

REFERENCES

[1] G. Kreisselmeier, R. Steinhauser. Systematic control design by optimizing a vector performance index. *Proc. IFAC Symp. on Computer Aided Design of Control System.* Zurich, Switzerland, 1979.

[2] G.P. McCormick. *Nonlinear Programming, Theory, Algorithm, and Applications.* John Wiley and Sons, New York, 1983.

[3] M.J.D. Powell. A method for nonlinear constraints in minimization problems. In: R. Fletcher, ed., *Optimization,* 283–298. Academic, London, 1969.

[4] M.R. Hestenes. Multiplier and gradient methods. *J. Optim. Theor. Appl.*, **4**(5): 303–320, 1969.

[5] A.B. Templeman, X.S. Li. A maximum entropy approach to constrained non-linear programming. *Eng. Optimization,* **12**: 191–205, 1987.

[6] X.S. Li. An aggregate function method for nonlinear programming. *Science in China,* (series A), **34**: 1467–1473, 1991.

[7] P. Hajela. Further developments in the controlled growth approach for optimal structural synthesis. ASME Paper 82-DET-62, 1982.

[8] P. Hajela. A look at two underutilized methods for optimum structural design. *Eng. Optimization,* **11**: 21–30, 1987.

[9] J. Sobieszczanski-Sobieski, B.B. James, A.R. Dovi. Structural optimization by multilevel decomposition. *AIAA J.*, **23**: 1775–1782, 1985.

[10] J. Sobieszczanski-Sobieski, A.R. Dovi, G.A. Wrenn. A new algorithm for general multiobjective optimization. *Proc. AIAA/ASME/ASCE/AHS 29th Structural Dynamics and Materials Conference,* Williamsburg, Virginia, Paper 88-2434, 1988.

[11] J.S. Arora, A.I. Chahande, J.K. Paeng. Multiplier methods for engineering optimization. *Int. J. Num. Meths. Eng.*, **32**: 1485–1525, 1991.

[12] W. Hock, K. Schittkowski, *Test Examples for Nonlinear Programming Codes* Springer-Verlag, Berlin–Heidelberg, 1981.

[13] J.S. Arora. *Introduction to Optimum Design,* McGraw–Hill Inc., 1989.

[14] R.T. Haftka, Z. Gurdal, M.P. Kamat. *Elements of Structural Optimization,* 2nd Edition. Kluwer Academic, 1990.

[15] T.K. Agarwal, O.O. Storaasli, D.T. Nguyen. A parallel-vector algorithm for rapid structural analysis on high-performance computers. *Proc. of AIAA/ASME/ASCE/AHS 31st SDM Conference,* Long Beach, CA April 2–4, 1990, AIAA paper No. 90-1149, 1990.