AIAA JOURNAL
Vol. 43, No. 8, August 2005

Genetic Algorithm for Mixed Integer Nonlinear Programming Problems Using Separate Constraint Approximations

Vladimir B. Gantovnik,* Zafer Gürdal,† Layne T. Watson,‡ and Christine M. Anderson-Cook§
Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

A new approach is described for reducing the number of the fitness and constraint function evaluations required by a genetic algorithm (GA) for optimization problems with mixed continuous and discrete design variables. The proposed additions to the GA make the search more effective and rapidly improve the fitness value from generation to generation. The additions involve memory as a function of both discrete and continuous design variables and multivariate approximation of the individual functions' responses in terms of several continuous design variables. The approximation is demonstrated for the minimum weight design of a composite cylindrical shell with grid stiffeners.

Nomenclature

c	=	counter
D	=	database at a given tree node
d	=	trust region radius, subscripted
E	=	real numbers
$f(v, x)$	=	fitness function
$g_j(v, x)$	=	constraint functions, $1 \leq j \leq p$
$g_0(v, x)$	=	objective function
n_e	=	number of exact analyses
n_i	=	number of requests for constraint evaluations
r	=	variation measure
$\boldsymbol{v}$	=	integer design vector of dimension k
$\boldsymbol{x}$	=	real design vector of dimension m
Z	=	integers
δ, ϵ	=	tolerance parameters
ζ	=	percent savings over standard genetic algorithm
ξ	=	percent savings due to approximations only

I. Introduction

MANY diverse applications are mathematically modeled in terms of mixed discrete–continuous variables. The optimization of such models is typically difficult due to their combinatorial nature and potential existence of multiple local minima in the search space. The engineering problems that contain integer, discrete, zero–one, and continuous design variables are often referred to as mixed integer nonlinear programming (MINLP) problems.

Genetic algorithms (GAs) are powerful tools for solving MINLP problems. These methods do not require gradient or Hessian information. However, to reach an optimal solution with a high degree of confidence, they typically require a large number of analyses during the optimization search. Performance of these methods is even more of an issue for problems that include continuous variables. Several studies have concentrated on improving the reliability and efficiency of GAs. Hybrid algorithms formed by the combination of a GA with local search methods provide increased performance when compared to a GA with a discrete encoding of real numbers or local search alone.[1]

Although GAs are robust global optimizers, they typically require a very large number of fitness function evaluations. Moreover, it is commonly observed that fitness values are frequently recalculated for some designs that appear repeatedly during the evolution of the population. This suggests an opportunity for performance improvement. To reduce the computational cost, the authors earlier used local improvements and memory for discrete problems so that information from previously analyzed design points is stored and utilized in later searches.[2,3] In the first approach, a memory binary tree was employed for a composite panel design problem to store pertinent information about laminate designs that have already been analyzed.[2] After the creation of a new population of designs, the tree structure is searched for either a design with identical stacking sequence or similar performance, such as a laminate with identical in-plane strains. Depending on the kind of information that can be retrieved from the tree, the analysis for a given laminate may be significantly reduced or may not be required at all. The second method is called local improvement.[3] This technique was applied to the problem of maximizing the buckling load of a rectangular laminated composite plate. The information about previously analyzed designs is used to construct an approximation to buckling load in the neighborhood of each member of the population of designs. After that, the approximations are used to search for improved designs in small discrete spaces around nominal designs. These two methods demonstrated substantial improvements in computational efficiency for purely discrete optimization problems. The implementation, however, was not suitable for handling continuous design variables.

New approaches have been proposed to overcome this shortcoming. In particular, a new version of GA has been recently developed,[4] consisting of memory as a function of both discrete and continuous design variables using spline[5] and multivariate[6] approximations of the constraint functions in terms of continuous design variables.

The work here proposes to enhance the efficiency and accuracy of the GA with memory using multivariate approximations of the objective and constraint functions individually instead of direct approximations of the overall fitness function. The primary motivation for the proposed improvements is the nature of the fitness function in constrained engineering design optimization problems. Because GAs are algorithms for unconstrained optimization, constraints are typically incorporated into the problem formulation by augmenting the objective function of the original problem with penalty terms associated with individual constraint violations. The resulting fitness function is usually highly nonlinear and discontinuous, which makes the multivariate approximation highly inaccurate unless a large number of exact function evaluations are performed. Because the individual response functions in many engineering problems are mostly smooth functions of the continuous variables (although they can be highly nonlinear), high-quality approximations to individual

Received 29 July 2003; revision received 21 March 2005; accepted for publication 4 April 2005. Copyright © 2005 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved. Copies of this paper may be made for personal or internal use, on condition that the copier pay the $10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 0001-1452/05 $10.00 in correspondence with the CCC.

*Graduate Research Assistant, Department of Engineering Science and Mechanics. Member AIAA.
†Professor, Departments of Aerospace and Ocean Engineering and Engineering Science and Mechanics; currently Aerospace Structures Chair, Delft University of Technology, 2629 HS Delft, The Netherlands. Associate Fellow AIAA.
‡Professor, Departments of Computer Science and Mathematics.
§Associate Professor, Department of Statistics.

functions can be constructed without requiring a large number of function evaluations. The proposed modification is, therefore, expected to improve the efficiency of the memory constructed in terms of the continuous variables. The paper presents the algorithmic implementation of the proposed memory scheme and demonstrates the efficiency of the proposed multivariate approximation procedure for the weight optimization of a lattice shell with laminated composite skins subjected to axial compressive load. The composite shell design problem is used as a demonstration problem, instead of a synthetic constrained optimization problem. Results are generated to demonstrate the advantages of the proposed improvements to a standard GA.

II. GA Package

A FORTRAN 90 GA framework that was designed in an earlier research effort was used for the composite laminate structure design.[7] This framework includes a module, encapsulating GA data structures, and a package of GA operators. The module and the package of operators result in what we call a standard GA. The proposed algorithm is incorporated within the GA framework to illustrate performance of the binary tree memory and multivariate approximation. An integer alphabet is used to code ply genes. The continuous variables represented by floating-point numbers had already been implemented in the GA framework data structure as geometry chromosomes.

III. Binary Tree Memory

A binary tree is a linked list structure in which each node may point to up to two other nodes. In a binary search tree, each left pointer points to nodes containing elements that are smaller than the element in the current node; each right pointer points to nodes containing elements that are greater than the element in the current node. The binary tree is used to store data pertinent to the design such as the design string and its associated fitness and constraint function values. A binary tree has several properties of great practical value, one of which is that the data can be retrieved, modified, and inserted relatively quickly. If the tree is perfectly balanced, the cost of inserting an element in a tree with n nodes is proportional to $\log_2 n$ steps, and rebalancing the tree after an insertion may take as little as several steps, but at most takes $\log_2 n$ steps. Thus, the total time is of the order of $\log_2 n$ (Ref. 8).

When the mechanisms of the GA operators are examined, it is observed that the diversity of a population trends to decrease as the algorithm runs longer. The fitness values for the same chromosomes are recalculated repeatedly, especially toward the end of the optimization process. If previously calculated fitness values can be efficiently saved and retrieved, computation time will decrease significantly. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. Algorithm 1 shows the pseudocode of the fitness function evaluation with the aid of the binary tree. After a new generation of designs is created by the genetic operations, the binary tree is searched for each new design. If the design is found, the fitness value is retrieved from the binary tree without conducting an analysis. Otherwise, the fitness is obtained based on an exact analysis. This new design and its fitness value are then inserted in the tree as a new node. The major improvement proposed here is to store not just the fitness value but the values of every function that can contribute to the computation of the fitness function.

Algorithm 1; Evaluation of fitness function using binary tree:
search for the given design in the binary tree;
if found **then**
  get the fitness function value from the binary tree;
else
  perform exact analysis;
end if

IV. Response Surface Approximations

The procedure just described works well for purely discrete optimization problems where designs are completely described by discrete strings. In the case of mixed optimization problems where designs include discrete and continuous variables, the solution becomes more complicated. If the continuous variables are also discretized into a fine discrete set, the possibility of creating a child design that has the same discrete and continuous parts as one of the earlier designs diminishes substantially. In the worst case, if the continuous design variables are represented as real numbers, which is the approach used in most recent research work, it may not be possible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless, and result in many exact analyses even if the real part of the new child is different from one of the earlier designs by a minute amount.

The main idea of the memory approach for problems with mixed discrete–continuous variables is to construct a response surface approximation for every constraint function as a function of the continuous variables using historical data values and to estimate from the stored data whenever appropriate. The memory in this case consists of two parts: a binary tree, which consists of the nodes that have different discrete parts of the design, and a storage part at each node that keeps the continuous values and the corresponding constraint functions' values. That is, each node contains several real arrays that store the continuous variables' values and their corresponding constraint functions' values. For the memory to be functional, it is necessary to have accumulated a sufficient number of designs with different continuous values for a particular discrete design point so that the approximations can be constructed. Naturally, some of the discrete nodes will not have more than a few designs with different continuous values. However, it is possible that as the GA search progresses, promising discrete parts will start appearing repeatedly with different continuous values. In this case, one will be able to construct good quality response surface approximations to the data.

The response surface approximation approach is an extension of the previous work by the authors in which a spline-based approach was used for only one continuous variable.[5] An evolving database of continuous variable points is used in the current work to construct multivariate response surface approximations at those discrete nodes that are processed frequently. The modified quadratic Shepard method is a local smoothing method used for the approximation of scattered data for the cases of two and three independent continuous design variables (see Refs. 9–11). It has been suggested that the modified quadratic Shepard method overcomes the drawbacks of a well-known interpolation scheme given by Shepard.[12] This method may be the best known among all scattered data interpolants for a general number of variables and has the advantage of numerical efficiency, stability, small storage requirements, and easy generalization to more than two independent variables. It, therefore, seems to be the most suitable candidate for handling a very large amount of data and for use in the case of a high number of independent variables.

In addition to building the multivariate approximations, it is important to assess accuracy of the multivariate approximation at new continuous points that have not been encountered before, so that a decision may be made either to accept the approximation or perform exact function evaluation. Based on the multivariate approximation, the proposed algorithm described by the following pseudocode is then used to decide when to retrieve the constraint function values from the approximations and when to do an exact analysis and add the new data point to the approximation database.

For the description of the pseudocode, let $\boldsymbol{v} \in Z^k$ be a k-dimensional integer design vector for the discrete space, $\boldsymbol{x} \in E^m$ a real m-dimensional design vector for the continuous variables, $g_0(\boldsymbol{v}, \boldsymbol{x})$ the corresponding objective function, $g_1(\boldsymbol{v}, \boldsymbol{x}), \ldots, g_p(\boldsymbol{v}, \boldsymbol{x})$ the corresponding constraint functions, and $f(\boldsymbol{v}, \boldsymbol{x})$ the corresponding fitness value of the individual defined in terms of the constraint functions and the objective function.

Define the symbol $d \in E$ with subscripts to be a real distance corresponding to a trust region radius about a specific point in the database. Let $D = (\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_p)$ contain the set of n observed exact analysis points and their corresponding information within a given discrete node, where

$$T_j = \left(\left\{ \left(I_{ji}, g_j\left(v, x^{(I_{ji})}\right), d_{ji} \right) \right\}_{i=1}^{c_j}, c_j, r_j \right)$$

is the data set associated with the jth constraint function, I_{ji} is the index pointing to the global design data set $\{x^{(i)}\}_{i=1}^{n}$, $g_j(\boldsymbol{v}, \boldsymbol{x}^{(I_{ji})})$ is

the value of the jth constraint, d_{ji} is the corresponding trust region radius, c_j is the counter indicating the number of points in the design data set corresponding to the jth constraint, and

$$r_j = \left| \max_{1 \le i \le c_j} \left\{ g_j\left(v, x^{(I_{ji})}\right) \right\} - \min_{1 \le i \le c_j} \left\{ g_j\left(v, x^{(I_{ji})}\right) \right\} \right|$$

is the difference between the current maximum and minimum values of the jth constraint. Components of the tuples D and T_j are denoted by subscripts, for example, $(T_j)_1$ is the first component of T_j. Finally, each node in the binary tree memory structure records a tuple of the form $(\boldsymbol{v}, D)$. The pseudocode, with comments in brackets, for processing a candidate individual $(\boldsymbol{v}, \boldsymbol{x})$ is defined by Algorithms 2 and 3:

$B(j, \boldsymbol{v}, \boldsymbol{x})$ is a Boolean function intended to provide the option of bypassing the algorithm's normal logic, if dictated by a priori knowledge about the function g_j or individual $(\boldsymbol{v}, \boldsymbol{x})$. The parameters $c_j^{\min}$ [minimum number of data points required to construct the approximation $s_j(\boldsymbol{x})$ to $g_j(\boldsymbol{v}, \boldsymbol{x})$, determined in the present work by the requirements of Shepard's algorithm] are defined separately for each constraint function, and their values are based on the function complexity and approximation method used for the constraint function. The algorithm uses three real user-specified parameters, d^0, δ, and ϵ, all indexed by j. The parameter $d^0 > 0$ is an upper bound on the trust region radius about each sample point $x^{(i)}$. The parameter δ is chosen to satisfy $0 < \delta < 1$ and, in higher dimensions, protects against large variations in f in unsampled directions. Finally, the

---

Algorithm 2; Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where $\boldsymbol{v}$ is not found in the tree:

if v is not found in the tree **then**
  evaluate $g_0(v, x), \ldots, g_p(v, x)$;
  evaluate $f(v, x)$; $n := 1$; $x^{(1)} := x$;
  for $j = 0$ to p **do**
    $c_j := 1$; $r_j := 0.0$; $I_{j1} := 1$; $d_{j1} := 0.0$;
    $T_j := (\{(I_{ji}, g_j(v, x^{(I_{ji})}), d_{ji})\}_{i=1}^{c_j}, c_j, r_j)$;
  end for
  $D := (\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_p)$; add a node corresponding to (v, D);
  return $f(v, x)$;
end if

Algorithm 3 (continuation of Algorithm 2); Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where $\boldsymbol{v}$ is found in the tree:

if v is found in the tree **then**
  for $j = 0$ to p **do**
    if $B(j, v, x)$ **then**
      evaluate $g_j(v, x)$;
    else
      if $c_j < c_j^{min}$ **then**
        [add new point x to database]
        evaluate $g_j(v, x)$; $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$; $c_j := c_j + 1$;
        $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(v, x), 0.0)\}$; $I_{jc_j} := n$;
        $r_j := |\max_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\} - \min_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\}|$;
      else
        construct $s_j(x)$ using the data in
        $\{(x^{(I_{ji})}, g_j(v, x^{(I_{ji})}))\}_{i=1}^{c_j}$;
        define k and d^* by
        $d^* := d_{jk} - \|x - x^{(I_{jk})}\| = \max_{1 \le i \le c_j}\{d_{ji} - \|x - x^{(I_{ji})}\|\}$;
        if $d^* \ge 0.0$ and $|g_j(v, x^{(I_{jk})}) - s_j(x)| < \delta_j r_j$ **then**
          [accept approximation as good enough]
          $g_j(v, x) := s_j(x)$;
        else
          evaluate $g_j(v, x)$;
          $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$;
          $c_j := c_j + 1$;
          [update trust region radius for x]
          if $|g_j(v, x) - s_j(x)| > \epsilon_j$ **then**
            $d_{jc_j} := 0.0$;
          else
            $d_{jc_j} := \min\{d_j^0, \|x - x^{(I_{jk})}\|\}$; $d_{jk} := d_{jc_j}$;
          end if
          [update node database with information for x]
          $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(v, x), d_{jc_j})\}$; $I_{jc_j} := n$;
          $r_j := |\max_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\} - \min_{1 \le i \le c_j}\{g_j(v, x^{(I_{ji})})\}|$;
        end if
      end if
    end if
  end for
  evaluate $f(v, x)$ using $g_0(v, x), \ldots, g_p(v, x)$;
  return $f(v, x)$;
end if

parameter $\epsilon > 0$ is the selected acceptable approximation accuracy and is based solely on engineering considerations.

V. Local Improvement

Local improvement is essentially an addition to improve the performance of the GA with memory binary tree and separate multivariate approximations. The effectiveness of local improvement was shown in previous work.[5,6] The values of the continuous variables at a given discrete node are either randomly assigned or obtained through the GA operations. If explicit multivariate approximations for all constraint functions are available at a given node, it is possible to generate good candidates for the continuous design variables for the next child at that node thorough local optimization rather than depending on random actions from the GA operators. After construction of all initial approximations $\tilde{g}_i$ of g_i, one can easily find the approximate fitness function $\tilde{f}(\boldsymbol{x})$ whose evaluations do not require any exact function evaluations. Next it is possible to find the design vector $\boldsymbol{x}^*$ that optimizes the approximate function $\tilde{f}(\boldsymbol{x})$ in some compact subset $\Omega \subset E^m$ containing the real data points $\boldsymbol{x}^{(i)}$ associated with that node. This optimal $\boldsymbol{x}^*$ vector is stored at the discrete node in addition to the rest of the database D. If, in future generations, a discrete node that has a stored $\boldsymbol{x}^*$ vector is reached through the GA operations on the discrete part $\boldsymbol{v}$ of the design, then, rather than performing crossover or mutation on the real part, $(\boldsymbol{v}, \boldsymbol{x}^*)$ is used as the child design for the next generation. This child design is treated like the other new designs in the child population to which Algorithm 2 is applied to avoid exact analysis. In an effort to reduce premature convergence, the local improvement procedure is applied with some probability.

VI. Design Optimization Problem

The design of a fiber-reinforced composite lattice shell with specified radius, length, and axial load level is considered as a demonstration problem for the procedure described. Such shells supported by a lattice have been considered as a replacement to solid shells, stiffened shells, and honeycomb structures.[13–15] Consider a lattice cylindrical shell loaded with compressive axial load P. In general, proper design would involve determination of rib parameters (dimension of cross section, material, spacing, and orientation angle φ), skin parameters (the number of layers, their materials, thicknesses, and orientation angles θ_k) that satisfy strength and stability constraints while minimizing the weight of the shell. Constraints considered include rib strength constraint g_1, skin strength constraint g_2, rib local buckling constraint g_3, and general buckling constraint g_4. All constraint equations are based on the lattice cylindrical shell model developed by Bunakov and Protasov,[16] Belousov and Bunakov,[17] and Bunakov.[18]

The mixed optimization problem considered here operates on three design variables $\boldsymbol{v}$, $\boldsymbol{x}_1$, and $\boldsymbol{x}_2$. The discrete variable is the stacking sequence encoding of the skins, $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k)$, where k is an implicit design variable dictated by the number of layers in the skin stacking sequence and $\boldsymbol{v}_i$ is the integer encoding for the ith fiber angle θ_i. We shall restrict our consideration to two continuous design variables, namely, the helical rib height, $\boldsymbol{x}_1 = h$, and the orientation angle of helical ribs with respect to the axial direction, $\boldsymbol{x}_2 = \varphi$. The optimization problem can be formulated as finding the stacking sequences of the skins, the angle of helical ribs, and the helical rib height to minimize the mass of the shell, g_0, and satisfy all constraints. The set of design variables is expressed as a vector $\boldsymbol{\tau} = (\boldsymbol{v}, \boldsymbol{x}_1, \boldsymbol{x}_2)$. The optimization problem can be written as

$$\min_{\tau} g_0(\boldsymbol{\tau}) \tag{1}$$

such that rib strength, skin strength, rib local buckling, and general shell buckling are, respectively,

$$g_1(\tau) \geq 0, \qquad g_2(\tau) \geq 0, \qquad g_3(\tau) \geq 0, \qquad g_4(\tau) \geq 0$$

and where

$$h \in [h_{\min}, h_{\max}], \qquad \varphi \in [\varphi_{\min}, \varphi_{\max}]$$

$$\theta_i \in \{0, \pm 45, 90\} \qquad (i = 1, \ldots, k)$$

$$k \in [k_{\min}, k_{\max}]$$

where $h_{\min}$ and $h_{\max}$ are the lower and upper bounds of the rib height, $\varphi_{\min}$ and $\varphi_{\max}$ are the lower and upper bounds of the angle of helical ribs, θ_i is the ply orientation angle (in degrees) in the ith skin ply, k is the total number of skin plies, and $k_{\min}$ and $k_{\max}$ are minimum and maximum possible values of k. The preceding problem may not be a realistic composite design formulation, but it is used instead of a completely artificial constrained optimization problem. A standard laminate optimization typically includes additional constraints such as ply contiguity, interlaminar stress, core strength, etc.

The constrained optimization problem is transformed into an unconstrained maximization problem using a penalty function approach. The critical constraint is defined as

$$g_{\text{cr}}(\tau) = \min_{1 \leq i \leq 4} \{g_i(\tau)\} \tag{2}$$

Note that min/max functions such as g_{cr} are typically nonsmooth and are much harder to approximate directly than are their constituent functions g_i. The fitness function f to be maximized is defined as

$$f(\tau) = \begin{cases} -g_0(\tau) + \alpha g_{\text{cr}}(\tau), & g_{\text{cr}}(\tau) \geq 0 \\ -g_0(\tau) + \beta g_{\text{cr}}(\tau), & g_{\text{cr}}(\tau) < 0 \end{cases} \tag{3}$$

where α is a bonus parameter and β is a user-defined penalty parameter.

VII. Results

A cylindrical lattice shell considered in this study is made of fiberglass–epoxy composite material with density $\rho = 1600$ kg/m^3. The shell radius and length are $R = 0.7$ m and $L = 1.8$ m, respectively. The specified axial compressive load is $P = 10^6$ N/m. The shell has external and internal skins made of T300/5208 graphite–epoxy unidirectional plies with basic ply thickness $h_0 = 0.125$ mm. The material properties of the skin plies are given in Table 1. The lattice shell has $\pm\varphi$ unidirectional helical ribs with elastic modulus $E = 45$ GPa and shear modulus $G = 1$ GPa. The ribs have initial rectangular cross sections of width $b = 4$ mm and height h. The helical rib spacing is $a = 40$ mm. The compressive strength of ribs is $\sigma^r_{\max} = 240$ MPa. The possible ranges for the design variables are given in Table 2.

A. GA Parameters

The values of the GA parameters used in the experiments are shown in Table 3. The GA stopping condition is a limit on the total

Table 1 Material properties of skin (T300/5208)

Parameter	Value
Stiffness	
E_1	181.0
E_2	10.3
G_{12}	7.17
ν_{12}	0.28
Strength	
X_t	1500.0
Y_t	40.0
X_c	1500.0
Y_c	246.0
S	68.0

Table 2 Ranges for design variables

Design variable	Range
$h \in [h_{\min}, h_{\max}]$	[0.001, 0.1], m
$\varphi \in [\varphi_{\min}, \varphi_{\max}]$	[5, 85], deg
$\theta_k, k = 1, n$	$\{0, \pm 45, 90\}$, deg
$n \in [n_{\min}, n_{\max}]$	[2, 14]

Table 3 GA parameters used in experiments

Parameter	Value
Selection type	Elitist
Maximum number of generations	25,000
Population size	20
Laminate chromosome length, λ	7
Crossover type	
Laminate chromosomes	One-point
Geometry chromosomes	Uniform
Probability of crossover, p_c	
Laminate chromosomes	1.0
Geometry chromosomes	1.0
Probability of mutation, p_m	
Laminate chromosomes	0.05
Geometry chromosomes	0.01
Bonus parameter α	0.0
Penalty parameter β	10.0

Table 4 Best-known optimal design using standard GA

Parameter	Value
$\bar{n}_e^0$	113,615
x_1	0.0100
x_2	62.8192
v^{a}	1100000
g_0	22.0723
j_{cr}	2
g_{cr}	0.0
f	−22.0723

[a]Integer encodings 1, 0 in v mean 0 deg and no ply, respectively.

Table 5 Efficiency of multivariate approximations

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, %	$\bar{\zeta}$, %	E
g_1	23,168	3383	84.88	97.02	$9.2915E{-}03$
g_2	23,168	3058	86.28	97.31	$1.0860E{-}05$
g_3	23,168	3931	82.39	96.54	$3.2553E{-}01$
g_4	23,168	3410	84.78	97.00	$3.8846E{-}04$

number of fitness function evaluations conducted by the standard GA, $(n_e^0)_{\max} = 500{,}000$. The best known global optimal design obtained by the standard GA is presented in Table 4. Table 4 gives the average number $\bar{n}_e^0$ of exact analyses of individuals from 10 runs, the continuous design variables $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the discrete design variable $\boldsymbol{v}$, the objective function value g_0, the critical constraint number j_{cr}, the critical constraint value g_{cr}, and fitness function value f. This design was obtained in an average of about 113,615 function evaluations by the standard GA.

B. Effect of GA Improvements

The results presented in this section focus on the ability of the proposed algorithm to save computational time during GA optimization with the multivariate approximation used as a memory device. The best design is identical to the results presented in Table 4 for the baseline algorithm. The performance of the GA with the multivariate approximation is presented in Table 5, which shows averages from 10 runs with the prescribed parameters $\epsilon = 0.01$, $\delta = 0.1$, and $d^0 = 0.5$. Table 5 shows the average number of attempts to evaluate constraint functions $\bar{n}_i$, the average number of exact analyses $\bar{n}_e$, the average percent savings $\bar{\xi}$ in terms of constraint function evaluations, the average percent savings $\bar{\zeta}$ in terms of constraint function evaluations as compared with the standard GA result reported in Table 4, and the mean absolute error E due to the approximations. The average percent savings $\bar{\xi}$ in terms of number of constraint function evaluations is defined by

$$\bar{\xi} = \frac{1}{r}\sum_{k=1}^{r}\left[1 - \frac{(n_e)_k}{(n_i)_k}\right] \times 100\% \tag{4}$$

where r is the number of the runs. The average percent savings $\bar{\zeta}$ in terms of constraint function evaluations as compared with the standard GA is defined by

$$\bar{\zeta} = \frac{1}{r}\sum_{k=1}^{r}\left(1 - \frac{(n_e)_k}{\bar{n}_e^0}\right) \times 100\% \tag{5}$$

The average mean absolute error E is defined as

$$E = \frac{1}{r}\sum_{k=1}^{r}\left(\frac{1}{n_s}\sum_{i=1}^{n_s}|g(x_i) - s(x_i)|\right)_k \tag{6}$$

where $n_s = n_i - n_e$ is the total number of acceptable approximate evaluations for the given constraint. This error is computed every time that the algorithm decides to extract an approximation of the constraint value without an exact analysis.

The results of the experiments in Table 5 show that the cost of the GA with continuous variables could be reduced up to 97% relative to the standard GA by using the approximation procedure. The results shown in Table 5 are for the same problem presented in Ref. 6 with very minor changes in the overall dimensions of the cylindrical shell. Compared to the improvements achieved in Ref. 6, in which the percent savings in terms of number of exact fitness evaluations compared to the standard GA was at most 86% (Table 9 of Ref. 6), the superiority of the approach proposed here is evident. Moreover, for the problem considered, the computation of the fitness functions is not very expensive in terms of CPU time. However, for realistic problems in which evaluation of the objective and/or constraint functions may require large finite element analysis models, the computation effort spent on evaluating the fitness function far exceeds that of the memory tree and approximation constructions. Therefore, the approach developed here has great potential for problems with expensive fitness functions.

Table 6 contains information about the problem design space. Table 6 includes the laminate chromosome length λ, the number of the possible alphabet elements q, the maximum number of nodes in the binary tree, that is, the number of all possible combinations, $N_{\max}$; the actual average number of nodes in the binary tree, $\bar{N}_t$; the average number of nodes containing at least one working approximation, $\bar{N}_a$; and the average number of design points used to construct approximations in all nodes of the binary tree, $\bar{N}_p$: The GA with memory converges fast and uses about 8% of all available binary tree nodes to obtain the optimal solution.

The mean absolute error E due to the approximation and the savings $\bar{\xi}$ in terms of the number of constraint evaluations for different values of the parameters ϵ and δ with $d^0 = 0.5$ for all constraint functions are shown in Table 7. It is possible to further enhance the performance of the algorithm by a more precise tuning of its parameters. Table 7 shows the expected trends; both average savings $\bar{\xi}$ and average absolute error E increase as either ϵ or δ increases.

Table 8 shows the performance of the GA with separate constraint approximations and local improvement. The local improvement procedure is a quasi-Newton optimization method. In an effort to reduce premature convergence of the GA, the local improvement procedure is applied with probability 0.5. As expected, the GA with approximations and local improvement converges faster in terms of number of fitness function evaluations than the GA with just the separate constraint approximations. Both algorithms with separate

Table 6 Design space for GA with multivariate approximation

Parameter	Value
λ	7
q	3
$N_{\max}$	3280
$\bar{N}_t$	272
$\bar{N}_a$	20
$\bar{N}_p$	8101

Table 7 Average percent savings $\bar{\xi}$ and average error of multivariate approximations E as functions of parameters ϵ and δ with $d^0 = 0.5$

	g_1		g_2		g_3		g_4	
δ	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E	$\bar{\xi}$, %	E
				$\epsilon = 0.001$				
0.1	42.23	$1.50E-03$	44.38	$3.14E-06$	32.15	$3.82E-03$	41.23	$1.21E-05$
0.5	45.54	$1.57E-03$	55.16	$3.26E-06$	34.15	$3.64E-02$	51.26	$1.68E-05$
1.0	60.06	$2.05E-03$	62.35	$4.28E-06$	60.12	$4.22E-02$	58.14	$1.95E-04$
				$\epsilon = 0.005$				
0.1	76.65	$2.36E-03$	64.29	$6.25E-06$	65.27	$4.26E-02$	69.26	$2.68E-04$
0.5	77.26	$5.26E-03$	68.25	$8.26E-06$	71.26	$4.59E-02$	76.65	$2.57E-04$
1.0	78.96	$6.27E-03$	70.15	$1.03E-05$	74.92	$6.26E-02$	82.64	$3.26E-04$
				$\epsilon = 0.01$				
0.1	84.88	$9.29E-03$	86.28	$1.09E-05$	82.39	$3.26E-01$	84.78	$3.88E-04$
0.5	85.16	$1.02E-02$	86.31	$3.49E-05$	82.69	$6.26E-01$	85.06	$6.57E-04$
1.0	85.21	$8.27E-02$	86.65	$5.32E-05$	82.98	$8.26E-01$	85.65	$8.27E-04$

Table 8 Efficiency of multivariate approximations and local improvement with $\epsilon = 0.01$, $\delta = 0.1$, and $d^0 = 0.5$

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, %	$\bar{\zeta}$, %	E
g_1	10,761	2094	80.54	98.16	$3.4354E-02$
g_2	10,761	1932	82.05	98.30	$1.0475E-02$
g_3	10,761	2346	78.20	97.94	$1.1426E-01$
g_4	10,761	2105	80.44	98.15	$2.4638E-02$

constraint approximations demonstrate good convergence in comparison with the standard GA and noticeably decrease the number of exact analyses. However, note that the mean absolute error is increased for the results with the local improvement procedure. A tradeoff analysis between the acceptable approximation accuracy and the overall modified GA performance is indicated.

VIII. Conclusions

Modifications of the standard GA to save previously computed fitness values provide significant performance improvement. A GA with memory along with multivariate approximations of the objective and constraint functions individually was applied to the problem of weight minimization of a lattice shell with mixed discrete and continuous design variables. The use of memory based on a binary tree for the discrete part of the design variables avoids repeating analyses of previously encountered designs. The multivariate approximation for continuous variables saves unnecessary exact analyses for points close to previous values.

Acknowledgments

This research was supported in part by Air Force Office of Scientific Research Grant F49620-99-1-0128 and National Science Foundation Grant DMS-9625968.

References

[1]Seront, G., and Bersini, H., "A New GA-Local Search Hybrid for Continuous Optimization Based on Multi-Level Single Linkage Clustering," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00)*, edited by L. D. Whitley, D. E. Goldberg, E. Cantú-Paz, L. Spector, I. C. Parmee, and H.-G. Beyer, Morgan Kaufmann, San Francisco, 2000, pp. 90–95.

[2]Kogiso, N., Watson, L. T., Gürdal, Z., and Haftka, R. T., "Genetic Algorithms with Local Improvement for Composite Laminate Design," *Structural Optimization*, Vol. 7, No. 4, 1994, pp. 207–218.

[3]Kogiso, N., Watson, L. T., Gürdal, Z., Haftka, R. T., and Nagendra, S., "Design of Composite Laminates by a Genetic Algorithm with Memory," *Mechanics of Composite Materials and Structures*, Vol. 1, No. 1, 1994, pp. 95–117.

[4]Gantovnik, V. B., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Panels," AIAA Paper 2002-1221, April 2002.

[5]Gantovnik, V. B., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Panels," *Composite Structures*, Vol. 58, No. 4, 2002, pp. 513–520.

[6]Gantovnik, V. B., Anderson-Cook, C. M., Gürdal, Z., and Watson, L. T., "A Genetic Algorithm with Memory for Mixed Discrete-Continuous Design Optimization," *Computers and Structures*, Vol. 81, No. 20, 2003, pp. 2003–2009.

[7]McMahon, M. T., Watson, L. T., Soremekun, G. A., Gürdal, Z., and Haftka, R. T., "A Fortran 90 Genetic Algorithm Module for Composite Laminate Structure Design," *Engineering with Computers*, Vol. 14, No. 3, 1998, pp. 260–273.

[8]Vowels, R. A., *Algorithms and Data Structures in F and Fortran*, Unicomp, Inc., Tucson, AZ, 1998, pp. 89–98.

[9]Renka, R. J., "Multivariate Interpolation of Large Sets of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 139–148.

[10]Renka, R. J., "Algorithm 660: QSHEP2D: Quadratic Shepard Method for Bivariate Interpolation of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 149, 150.

[11]Renka, R. J., "Algorithm 661: QSHEP3D: Quadratic Shepard Method for Trivariate Interpolation of Scattered Data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, 1988, pp. 151, 152.

[12]Shepard, D., "A Two-dimensional Interpolation Function for Irregularly Spaced Data," *Proceedings of the 23rd National Conference*, Association for Computing Machinery, Brandon/Systems Press, Princeton, NJ, 1968, pp. 517–523.

[13]Vasiliev, V. V., and Lopatin, A. V., "Theory of Lattice and Stiffened Composite Shells," *Mechanics of Composite Materials*, edited by Y. M. Tarnopolskii, Zinatne, Riga, Latvia, 1992, pp. 82–88 (in Russian).

[14]Vasiliev, V. V., Barynin, V. A., and Rasin, A. F., "Anisogrid Lattice Structures—Survey of Development and Application," *Composite Structures*, Vol. 54, No. 2–3, 2001, pp. 361–370.

[15]Slinchenko, D., and Verijenko, V. E., "Structural Analysis of Composite Lattice Shells of Revolution on the Basis of Smearing Stiffness," *Composite Structures*, Vol. 54, No. 2–3, 2001, pp. 341–348.

[16]Bunakov, V. A., and Protasov, V. D., "Cylindrical Lattice Composite Shells," *Mechanics of Composite Materials*, Vol. 25, No. 6, 1989, pp. 1046–1053 (in Russian).

[17]Belousov, P. S., and Bunakov, V. A., "Bending of Cylindrical Lattice Composite Shells," *Mechanics of Composite Materials*, Vol. 28, No. 2, 1992, pp. 225–231 (in Russian).

[18]Bunakov, V. A., "Design of Axially Compressed Composite Cylindrical Shells with Lattice Stiffeners," *Optimal Design*, edited by V. V. Vasiliev and Z. Gürdal, Technomic, Lancaster, PA, 1999, pp. 207–246.

E. Livne
Associate Editor