A 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, VA, 24061

Abstract

This paper describes a new approach 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.

Introduction

There are many diverse applications that are mathematically modelled 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 which contain integer, discrete, zero-one, and continuous design variables are often referred to as mixed integer nonlinear programming (MINLP) problems.

[*]Graduate Research Assistant, Department of Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061, Student Member AIAA

[†]Professor, Departments of Aerospace and Ocean Engineering, and Engineering Science and Mechanics, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061, Associate Fellow AIAA

[‡]Professor, Departments of Computer Science and Mathematics, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061

[§]Associate Professor, Department of Statistics, Virginia Polytechnic Institute and State University, Blacksburg, VA, 24061

Genetic algorithms (GA) 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. In order 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. Since 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. Since 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 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 than a synthetic constrained optimization problem. Results are generated to demonstrate the advantages of the proposed improvements to a standard genetic algorithm.

Genetic algorithm 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 genetic algorithm. 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.

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 $\log_2 n$.[8]

By examining the mechanisms of the GA operators, 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 towards 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 pseudo code of the fitness function evaluation with the aid of the binary tree. After a new generation of

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
```

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.*

Response surface approximations

The procedure described above works well for purely discrete optimization problems where designs are completely described by discrete strings. In 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 by 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 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. In order 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 where 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.[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 pseudo code 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 pseudo code, let $v \in Z^k$ be a k-dimensional integer design

Algorithm 2 Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where 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**
    $c_j := 1$; $r_j := 0.0$; $I_{j1} := 1$; $d_{j1} := 0.0$;
    $T_j := \Big(\Big\{\big(I_{ji}, g_j(v, x^{(I_{ji})}), d_{ji}\big)\Big\}_{i=1}^{c_j}, c_j, r_j\Big)$;
  end for
  $D := \big(\{x^{(i)}\}_{i=1}^{n}, T_0, T_1, \ldots, T_p\big)$; add a node corresponding to (v, D);
  return $f(v,x)$;
end if

vector for the discrete space, $x \in E^m$ a real m-dimensional design vector for the continuous variables, $g_0(v,x)$ the corresponding objective function, $g_1(v,x),\ldots,g_p(v,x)$ the corresponding constraint functions, and $f\big(v,x\big)$ the corresponding fitness value of the individual defined in terms of the constraint functions and the objective function.

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

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

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

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

is the difference between the current maximum and minimum values of the jth constraint. Finally, each node in the binary tree memory structure records a tuple of the form (v, D). The pseudo code for processing a candidate individual (v,x) is defined by Algorithms 2 and 3. The parameters c_j^{min} 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°, δ, and ϵ, all indented by j. The parameter $d^\circ > 0$ 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

Algorithm 3 (Continuation of Algorithm 2) Evaluation of fitness function using binary tree and m-dimensional approximations to the constraint functions, case where 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**
        evaluate $g_j(v, x)$; $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$; $c_j := c_j + 1$;
        $(T_j)_1 := (T_j)_1 \cup \{(n, g_j(v, x), 0.0)\}$; $I_{jc_j} := n$;
        $r_j := \left| \max\limits_{1 \le i \le c_j} \left\{ g_j(v, x^{(I_{ji})}) \right\} - \min\limits_{1 \le i \le c_j} \left\{ g_j(v, x^{(I_{ji})}) \right\} \right|$;
      else
        construct $s_j(x)$ using the data in
        $\left\{ \left(x^{(I_{ji})}, g_j\left(v, x^{(I_{ji})}\right) \right) \right\}_{i=1}^{c_j}$;
        define k and d^* by
        $d^* := d_{jk} - \left\| x - x^{(I_{jk})} \right\| = \max\limits_{1 \le i \le c_j} \left\{ d_{ji} - \left\| x - x^{(I_{ji})} \right\| \right\}$;
        if $d^* \ge 0.0$ and $\left| g_j\left(v, x^{(I_{jk})}\right) - s_j(x) \right| < \delta_j \, r_j$ **then**
          $g_j(v, x) := s_j(x)$;
        else
          evaluate $g_j(v, x)$;
          $D_1 := D_1 \cup \{x\}$; $n := |D_1|$; $x^{(n)} := x$;
          $c_j := c_j + 1$;
          if $|g_j(v, x) - s_j(x)| > \epsilon_j$ **then**
            $d_{jc_j} := 0.0$;
          else
            $d_{jc_j} := \min\left\{ d_j^{\circ}, \left\| x - x^{(I_{jk})} \right\| \right\}$; $d_{jk} := d_{jc_j}$;
          end if
          $(T_j)_1 := (T_j)_1 \cup \left\{ \left(n, g_j(v, x), d_{jc_j} \right) \right\}$; $I_{jc_j} := n$;
          $r_j := \left| \max\limits_{1 \le i \le c_j} \left\{ g_j(v, x^{(I_{ji})}) \right\} - \min\limits_{1 \le i \le c_j} \left\{ g_j(v, x^{(I_{ji})}) \right\} \right|$;
        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

large variations in f in unsampled directions. Finally, the parameter $\epsilon > 0$ is the selected acceptable approximation accuracy, and is solely based on engineering considerations.

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 $\tilde{f}(x)$ whose evaluations do not require any exact function evaluations. Next it is possible to find the design vector x^* that optimizes the approximate function $\tilde{f}(x)$ in some compact subset $\Omega \subset E^m$ containing the real data points $x^{(i)}$ associated with that node. This optimal 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 x^* vector is reached through the GA operations on the discrete part v of the design, then, rather than performing crossover or mutation on the real part, (v, 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.

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.[16–18]

The mixed optimization problem considered here operates on three design variables v, x_1, and x_2. The discrete variable is the stacking sequence of the skins, $v = \{\theta_1, \ldots, \theta_n\}$, where n

is an implicit design variable dictated by the number of layers in the skin stacking sequence. We shall restrict our consideration to two continuous design variables, namely, the helical rib height, $x_1 = H$, and the orientation angle of helical ribs with respect to the axial direction, $x_2 = \varphi$. The optimization problem can be formulated as finding the stacking sequences of the skins, the angle of helical ribs, and the helical rib height in order to minimize the mass of the shell, g_0, and satisfy all constraints. The set of design variables is expressed as a vector $\tau = (v, x_1, x_2)$. The optimization problem can be written as

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

such that

$$\begin{aligned}
&g_1(\tau) \geq 0 \quad \text{(rib strength)},\\
&g_2(\tau) \geq 0 \quad \text{(skin strength)},\\
&g_3(\tau) \geq 0 \quad \text{(rib local buckling)},\\
&g_4(\tau) \geq 0 \quad \text{(general shell buckling)},\\
&H \in [H_{min}, H_{max}],\\
&\varphi \in [\varphi_{min}, \varphi_{max}],\\
&\theta_k \in \{0^\circ, \pm 45^\circ, 90^\circ\}, \quad (k = 1, n),\\
&n \in [n_{min}, n_{max}],
\end{aligned}$$

where H_{min} and H_{max} are the lower and upper bounds of the rib height; φ_{min} and φ_{max} are the lower and upper bounds of the angle of helical ribs, θ_k is the ply orientation angle in the kth skin ply, n is the total number of skin plies, n_{min} and n_{max} are minimum and maximum possible values of n. The above problem may not be a realistic composite design formulation, but 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

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

The fitness function f to be maximized is defined as

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

where α is a bonus parameter, β is a user defined penalty parameter.

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 $\bar{\sigma}^r = 240$ MPa. The possible ranges for the design variables are given in Table 2.

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 number of fitness function evaluations conducted by the standard GA, $(n_e^\circ)_{max} = 500000$. The best known global optimal design obtained by the standard GA is presented in Table 4. The table gives the average number of exact analyses from ten runs of individuals ($\bar{n}_e^\circ$), the continuous design variables (x_1, x_2), the discrete design variable (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 113615 function evaluations by the standard GA.

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 previously 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 ten runs with the prescribed parameters $\epsilon = 0.01$, $\delta = 0.1$, $d^\circ = 0.5$. This table 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

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

The average mean absolute error E is defined as

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

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 show that the cost of the GA with continuous variables could be reduced up to 90% relative to the standard GA by using the approximation procedure. 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 developed approach 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, i.e., 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 ($\bar{N}_a$), the average number of design points used to construct approximations in all nodes of the binary tree ($\bar{N}_p$). The genetic algorithm 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 $d^\circ = 0.5$ 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 constraint approximations demonstrate good convergence in comparison with the standard GA, and noticeably decrease the number of exact analyses. However, it should be noted that the mean absolute error is increased for the results with the local improvement procedure. A trade-off analysis between the acceptable approximation accuracy and the overall modified GA performance is indicated.

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.

Acknowledgements

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 Optimization Based on Multi Level Single Linkage Clustering," July 2000, Genetic and Evolutionary Computation Conference (GECCO-2000).

[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," April 2002, 43rd AIAA/ ASME/ ASCE/ AHS/ ASC Structures, Structural Dynamics, and Materials Conference, AIAA Paper No. 2002-1221.

[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, 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," Sept. 2002, 9th AIAA/ ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA Paper No. 2002-5431.

[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," *Eng. Computers*, Vol. 14, 1998, pp. 260–273.

[8]Vowels, R. A., *Algorithms and Data Structures in F and Fortran*, Unicomp, Inc, Tucson, Arizona, 1998.

[9]Renka, R. J., "Multivariate interpolation of large sets of scattered data," *ACM Transactions on Mathematical Software*, Vol. 14, No. 2, June 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, June 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, June 1988, pp. 151–152.

[12]Shepard, D., "A two-dimensional interpolation function for irregularly spaced data," *Proceedings of the 23rd National Conference, ACM*, 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, 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, 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, 2001, pp. 341–348.

[16]Bunakov, V. A. and Protasov, V. D., "Cylindrical lattice composite shells," *Mechanics of Composite Materials*, , 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*, , 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 Publishing Co., Lancaster, PA, 1999, pp. 207–246.

List of Table Captions

Table 1: The material properties of the skin (T300/5208)

Table 2: Ranges for the design variables

Table 3: GA parameters used in the experiments

Table 4: The best known optimal design using standard GA

Table 5: The efficiency of the multivariate approximations

Table 6: Design space for GA with multivariate approximation

Table 7: The average percent savings ($\bar{\xi}$) and the average error of the multivariate approximations (E) as functions of the parameters ϵ and δ with $d^{\circ} = 0.5$

Table 8: The efficiency of the multivariate approximations and local improvement with $\epsilon = 0.01$, $\delta = 0.1$, and $d^{\circ} = 0.5$

Stiffness parameters, GPa				Strength parameters, MPa				
E_1	E_2	G_{12}	ν_{12}	X_t	Y_t	X_c	Y_c	S
181.0	10.3	7.17	0.28	1500.0	40.0	1500.0	246.0	68.0

TABLE 1:

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

Table 2:

Parameter	Value
Selection type	elitist
Maximum number of generations	25000
Population size	20
Laminate chromosome length (λ)	7
Crossover type:	
• for laminate chromosomes	one-point
• for geometry chromosomes	uniform
Probability of crossover (p_c):	
• for laminate chromosomes	1.0
• for geometry chromosomes	1.0
Probability of mutation (p_m):	
• for laminate chromosomes	0.05
• for geometry chromosomes	0.01
Bonus parameter α	0.0
Penalty parameter β	10.0

TABLE 3:

$\bar{n}_e^\circ$	x_1	x_2	v	g_0	j_{cr}	g_{cr}	f
113615	0.0100	62.8192	1100000	22.0723	2	0.0	-22.0723

Table 4:

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, (%)	$\bar{\zeta}$, (%)	E
g_1	23168	3383	84.88	97.02	9.2915E-03
g_2	23168	3058	86.28	97.31	1.0860E-05
g_3	23168	3931	82.39	96.54	3.2553E-01
g_4	23168	3410	84.78	97.00	3.8846E-04

TABLE 5:

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

Table 6:

ϵ	δ	g_1		g_2		g_3		g_4	
		$\bar{\xi}$, (%)	E	$\bar{\xi}$, (%)	E	$\bar{\xi}$, (%)	E	$\bar{\xi}$, (%)	E
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
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
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 7:

g	$\bar{n}_i$	$\bar{n}_e$	$\bar{\xi}$, (%)	$\bar{\zeta}$, (%)	E
g_1	10761	2094	80.54	98.16	3.4354E-02
g_2	10761	1932	82.05	98.30	1.0475E-02
g_3	10761	2346	78.20	97.94	1.1426E-01
g_4	10761	2105	80.44	98.15	2.4638E-02

TABLE 8: