A Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Panels

Vladimir B. Gantovnik[a], **Zafer Gürdal**[a], **and Layne T. Watson**[b]

[a] *Virginia Polytechnic Institute and State University, Department of Engineering Science and Mechanics, Blacksburg, VA 24061-0219 USA*

[b] *Virginia Polytechnic Institute and State University, Departments of Computer Science and Mathematics, Blacksburg, VA 24061-0106 USA*

Abstract

This paper is concerned with augmenting genetic algorithms (GAs) to include memory for continuous variables, and applying this to stacking sequence design of laminated sandwich composite panels that involves both discrete variables and a continuous design variable. The term "memory" implies preserving data from previously analyzed designs. A balanced binary tree with nodes corresponding to discrete designs renders efficient access to the memory. For those discrete designs that occur frequently, an evolving database of continuous variable values is used to construct a spline approximation to the fitness as a function of the single continuous variable. The approximation is then used to decide when to retrieve the fitness function value from the spline and when to do an exact analysis to add a new data point for the spline. With the spline approximation in place, it is also possible to use the best solution of the approximation as a local improvement during the optimization process. The demonstration problem chosen is the stacking sequence optimization of a sandwich plate with composite face sheets for weight minimization subject to strength and buckling constraints. Comparisons are made between the cases with and without the binary tree and spline interpolation added to a standard genetic algorithm. Reduced computational cost and increased performance index of a genetic algorithm with these changes are demonstrated.

Keywords: Genetic Algorithm, Composite Panel Structure, Spline Approximation

1. Introduction

Traditionally, the problem of composite laminate stacking sequence optimization has been defined as a continuous design problem and solved using gradient-based techniques [1]. However, because of manufacturing considerations, the orientations of the fibers in plies are typically confined within a discrete set, for example, $(0°, +45°, -45°, 90°)$. The stacking sequence optimization in this case could be formulated as an integer-programming problem [2]. Recent studies have shown that genetic algorithms (GAs) are highly suitable for the solution of the composite laminate design problems with discrete design variables [3], [4]. The solution of such problems by GA is possible, because the method does not require gradient or Hessian information. A genetic algorithm is a powerful technique for search and optimization problems with discrete variables, and is therefore particularly useful for optimization of composite laminates. However, to reach an optimal solution with a high degree of confidence, it typically requires a large number of analyses during the optimization search. Performance of GAs 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 [5]. In order to reduce the computational cost, two of the authors earlier used local improvements and memory so that information from previously analyzed design points is utilized during a search [6],

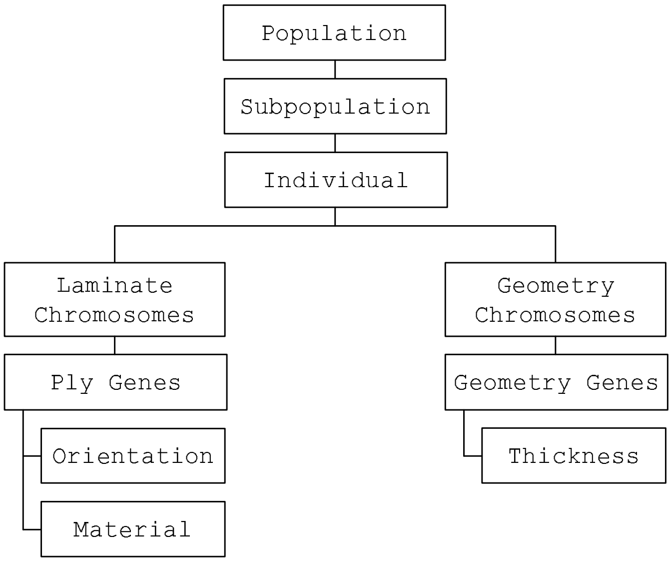

Fig. 1. The structure of a population.

[7]. In the first method a memory binary tree was employed to store pertinent information about laminate designs that have already been analyzed [6]. 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 [7]. 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.

The objective of the present work is to find a suitable algorithm for a GA with memory that can work with discrete and continuous variables simultaneously. A local memory for the continuous part of the design space based on spline approximation is proposed for problems with a single continuous variable. The efficiency of the proposed spline based procedure, as well as the use of memory for a GA that can handle continuous variables, are investigated for the weight optimization of a sandwich plate with composite face sheets subjected to strength and buckling constraints.

2. Genetic algorithm package

A state-of-the-art Fortran 90 GA framework that was designed in an earlier research effort was used for the composite laminate structure design [8]. 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 as a sample test program that illustrates performance of the binary tree memory and spline interpolation. An integer alphabet is used to code ply genes. It should be pointed out that the continuous variables had already been implemented in the GA data structure as geometry chromosomes, represented directly as real numbers and not discrete binary approximations. Figure 1 from [8] shows the data structure of a population for the most general case of composite structure design. This data structure supports advanced features such as migration between independently evolving subpopulations, use of multiple materials for designing hybrid laminates, and multiple laminated substructures represented by multiple laminate chromosomes. In general, the laminate chromosomes are for discrete variables, and the geometry chromosomes are for continuous variables. Migration is not used here, however, so there is just one subpopulation identical to the population, and there is just one laminate chromosome with one material type. Here only the ply orientation genes are used for the stacking sequence definition and a single geometry gene is used for the single continuous variable.

3. 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, as shown in Figure 2. 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 of 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 $\log_2 n$ [9].

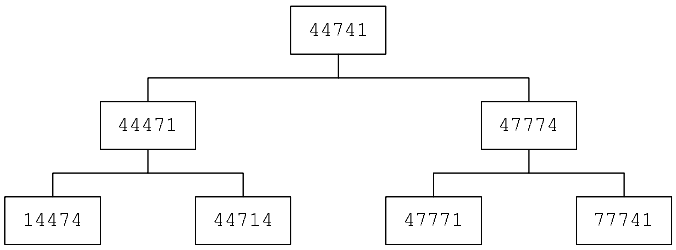

Fig. 2. An example of a binary tree.

In the standard genetic algorithm, a new population may contain designs that have already been encountered in the previous generations, especially towards the end of the optimization process. The memory procedure eliminates the possibility of repeating an analysis that could be expensive. The binary tree, keyed on the discrete design string encodings, is used to store data pertinent to the design such as the design string and the fitness and constraint functions. Figure 3 shows the pseudo code of the fitness function evaluation with the aid of the binary tree.

Procedure Evaluation of the fitness function using a binary tree
begin
    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
end

Fig. 3. Evaluation of fitness function using binary tree.

Procedure Evaluation of the fitness function using a binary tree and spline
begin
get a candidate individual (v, x) for insertion in the tree;
search for v of the given design in the binary tree;
if v is not found in the tree, **then**
  evaluate $\Phi(v, x)$;
  add a node corresponding to $(v, x, \Phi(v, x), 0)$;
  return with value $\Phi(v, x)$ to the GA;
else
  let (v, x_i, Φ_i, d_i), $1 \leq i \leq n$, be all the node data matching v;
  compute a spline S interpolating the data (x_i, Φ_i),$1 \leq i \leq n$;
  evaluate the spline $S(x)$;
  if $\max_{1\leq i \leq n}\{d_i - |x - x_i|\} \geq 0$, **then**
    return with the value $S(x)$ to the GA;
  else
    evaluate $\Phi(v, x)$;
    if $|\Phi(v, x) - S(x)| > \epsilon$, **then**
      add a node corresponding to $(v, x, \Phi(v, x), 0)$;
      return with the value $\Phi(v, x)$ to the GA;
    else
      define k and d by $|x - x_k| = \min_{1\leq i \leq n} |x - x_i|, d = \min\{d_0, |x - x_k|\}$;
      change the data at node (v, x_k, Φ_k, d_k) to (v, x_k, Φ_k, d);
      add a node corresponding to $(v, x, \Phi(v, x), d)$;
      return with the value $\Phi(v, x)$ to the GA;
    end if
  end if
end if
end

Fig. 4. Evaluation of fitness function using binary tree and spline.

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 data are then inserted in the tree as a new node.

4. Spline interpolation

The procedure described above works well for designs that are completely described by discrete strings. In case of designs that include a continuous variable, the solution is more complicated. If the continuous variable is 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 variable is represented as a real number, which is the approach used by most recent research work, it may be impossible to create a child design that has the exact same real part as one of the parents, rendering the binary tree memory useless.

The main idea of the approach proposed in this paper is to construct approximations for the fitness function as a function of the continuous variable using a spline function fitted to the historical data, and interpolate from the stored data whenever possible. 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 associated fitness function. That is, each node contains a real array that stores the continuous variable and its corresponding value of the fitness function. In order for the memory to be functional, it is necessary to have accumulated enough designs with different continuous values for a particular discrete design part. Naturally, not all the nodes will have more than a few designs with different continuous values. However, it is possible that as the evolution progresses good discrete parts will start appearing again and again with different continuous values. In this case, one will be able to construct a good quality spline interpolant to the data.

In addition to building a spline approximation it is important to develop a scheme in which accuracy of the spline approximation at a new continuous point may be assessed, so that a decision may be made either to accept the approximation or perform exact function evaluation. In the proposed approach for all the design points with identical discrete variables and different continuous variable values, a measure of distance within which a good quality approximation may be obtained is used to make the decision. Based on the stored information, the procedure described by pseudo code in Figure 4 decides whether an exact analysis should be performed and stored in the tree, or to extract an approximation of the fitness value without an exact analysis.

For the description of the pseudo code, let v be a discrete vector variable, x a scalar real variable, $\Phi(v, x)$ the fitness of the individual defined by (v, x), and d a real number representing the radius of an interval within which a good quality approximation is possible. Each node in the binary tree memory structure records the data $(v, \{x_i, \Phi(v, x_i), d_i\}_{i=1}^{n})$, where n is the number of times the design is evaluated with different values of the real design variable corresponding to the same discrete part. For the first three values of the continuous variable at a given discrete branch, the points are automatically evaluated and entered into the spline database with radius values d of zero. Since the initial spline points will have zero radius, many of the additional points x will not satisfy the proximity condition $\max_{1 \le i \le n}\{d_i - |x - x_i|\} \ge 0$ initially, and therefore their fitness $\Phi(v, x)$ will be computed exactly. It is possible to add the new points either with a zero radius or a positive computed value for the radius. If the actual value of the fitness $\Phi(v, x)$ and the spline approximation $S(x)$ are within acceptable error limit ϵ, then the radius value d for the new point and the closest point x_k to the new point are both changed to the distance between the two points, $d = \min\{d_0, |x - x_k|\}$, and the new point is added to the spline database with this computed value. Otherwise, the new point is added to the database with a zero radius value (see Fig. 4, for the pseudo code). The parameter d_0 is an upper bound for the distance d that protects the algorithm against large errors between exact and spline function values.

The approximation of the fitness function is based on spline interpolation of order less than or equal to four with the "not-a-knot" boundary condition. The "not-a-knot" condition is used when no information about the end point derivatives of an approximated function is known. The subroutine CUBSPL from the Fortran 90 version of Carl de Boor's cubic spline package was used to construct the spline approximations [10].

5. Local improvement

Local improvement is essentially an addition to improve the performance of the GA with spline interpolation. The value of the continuous variable at a given discrete node is either randomly assigned or obtained through a crossover operation. If an explicit spline approximation is available at a given node, it is possible to use a spline interpolant to generate a good candidate for the continuous variable for the next point at that node rather than depend on random action from the crossover operator. That is, after construction of an initial approximation $S(x)$ of $\Phi(v, x)$ based on the objective function values obtained at the design sites, one can easily find the point x^* that minimizes the spline function $S(x)$ in the range $[x_1, x_n]$. This optimal x^* value is stored at the discrete node in addition to the rest of the spline database. If, in future generations, a discrete node that has a stored x^* value is reached through the crossover operation on the discrete part v of the design, then, rather than performing crossover on the real part, (v, x^*) is used as the child design for the next generation. This child design will then be treated like the other new designs in the child population and will be checked if an approximation to it can be used without exactly evaluating it.

6. Analysis

The optimum design of a honeycomb sandwich construction for strength and buckling constraints is treated in this paper. The core depth, which is a real variable, the ply orientation angles and the number of layers, which are discrete variables, are taken as the design variables. Consider a honeycomb sandwich plate with length a, width b, face ply thickness t_p, core depth t_h, core cell wall thickness t_c, and cell size diameter t_d. When such a structure is subjected to biaxial compression, there are several modes of sandwich plate failure. One obvious way is overstressing. The strength of faces is determined by the maximum strain criterion. The plate is assumed to fail if any of the ply strains, $\{\epsilon_1^i, \epsilon_2^i, \gamma_{12}^i\}_{i=1}^n$, exceeds its allowable value. The critical strength failure load factor is defined as

$$\lambda_s = \min_{1 \le i \le n} \left[\min\left(\frac{\epsilon_1^*}{\epsilon_1^i}, \frac{\epsilon_2^*}{\epsilon_1^i}, \frac{\gamma_{12}^*}{\gamma_{12}^i} \right) \right], \tag{1}$$

where $\{\epsilon_1^*, \epsilon_2^*, \gamma_{12}^*\}$ are the ultimate allowable strains. The strength constraint is represented by

$$G_s = \lambda_s - 1.0 \ge 0. \tag{2}$$

Before overstressing, it is also possible that the structure can buckle. The reliability for the optimum design of the honeycomb sandwich construction is enhanced by adding several buckling constraints. Three modes of instability are considered as buckling constraints: overall buckling, face wrinkling, and face dimpling.

The overall buckling constraint for a honeycomb sandwich panel is represented by

$$G_b = \min\left[\frac{P_{cr}^*}{N_x}, \frac{P_{cr}^*}{N_y} \right] - 1.0 \ge 0, \tag{3}$$

where P_{cr}^* is the lowest value of the critical buckling load.

The computation of the buckling load is based on the mathematical model proposed by Hwu and Hu [11]. A closed-form solution of the buckling load was used for the composite sandwich plate consisting of cross-ply symmetric laminate faces with all edges simply supported:

$$P_{cr} = \frac{(1+\eta)P_0}{[1 + k(a/b)^2(n/m)^2](1 + \eta_x + \eta_y + \eta_0)}, \tag{4}$$

where

$$\eta = \frac{\pi^2 D^*}{t_h} \left[\frac{1}{G_{yz}} \left(\frac{m}{a} \right)^2 + \frac{1}{G_{xz}} \left(\frac{n}{b} \right)^2 \right]; \quad \eta_0 = \frac{D^* P_0 m^2 n^2}{a^2 t_h^2 G_{xz} G_{yz}};$$

$$\eta_x = \frac{\pi^2}{t_h G_{xz}} \left[D_{11} \left(\frac{m}{a} \right)^2 + D_{66} \left(\frac{n}{b} \right)^2 \right];$$

$$\eta_y = \frac{\pi^2}{t_h G_{yz}} \left[D_{66} \left(\frac{m}{a} \right)^2 + D_{22} \left(\frac{n}{b} \right)^2 \right];$$

$$P_0 = \frac{\pi^2 a^2}{m^2} \left[D_{11} \left(\frac{m}{a} \right)^4 + 2(D_{12} + 2D_{66}) \left(\frac{mn}{ab} \right)^2 + D_{22} \left(\frac{n}{b} \right)^4 \right];$$

$$D^* P_0 = \frac{\pi^2 a^2}{m^2} \left[D_{11} D_{66} \left(\frac{m}{a} \right)^4 + (D_{11} D_{22} - D_{12}(D_{12} + 2D_{66})) \left(\frac{mn}{ab} \right)^2 + D_{22} D_{66} \left(\frac{n}{b} \right)^4 \right];$$

$k = N_y/N_x$ is the loading ratio; m and n are the number of buckling half-waves within the deformed surface; D_{ij} are the bending stiffnesses of the laminate with respect to the midsurface of the core; G_{xz} and G_{yz} are the transverse shear moduli of the core material in the $(x-z)$ and $(y-z)$ planes.

Face wrinkling may occur across many cells of the honeycomb core. The face wrinkling constraint can be expressed as

$$G_{fw} = \min\left[\frac{\bar{N}_x^{fw}}{N_x}, \frac{\bar{N}_y^{fw}}{N_y}\right] - 1.0 \geq 0, \tag{5}$$

where

$$\bar{N}_x^{fw} = \left[\left(\frac{1}{N_x(1+G_s)}\right)^3 + \left(\frac{1}{N_x^{fw}}\right)^3\right]^{-\frac{1}{3}};$$
$$\bar{N}_y^{fw} = \left[\left(\frac{1}{N_y(1+G_s)}\right)^3 + \left(\frac{1}{N_y^{fw}}\right)^3\right]^{-\frac{1}{3}};$$
$$N_x^{fw} = \left[\left(D_{11} - \frac{D_{12}^2}{D_{22}}\right)E_c G_{xz}\right]^{\frac{1}{3}}; \quad N_y^{fw} = \left[\left(D_{22} - \frac{D_{12}^2}{D_{11}}\right)E_c G_{yz}\right]^{\frac{1}{3}}.$$

In honeycomb core sandwiches, a third type of instability, which is generally referred to as the face dimpling, may occur because the face sheet over one cell of the honeycomb can buckle as a small plate supported by the cell walls. The face dimpling constraint can be written as

$$G_{fd} = \min\left[\frac{\bar{N}_x^{fd}}{N_x}, \frac{\bar{N}_y^{fd}}{N_y}\right] - 1.0 \geq 0, \tag{6}$$

where

$$\bar{N}_x^{fd} = \left[\left(\frac{1}{N_x(1+G_s)}\right)^3 + \left(\frac{1}{N_x^{fd}}\right)^3\right]^{-\frac{1}{3}};$$
$$\bar{N}_y^{fd} = \left[\left(\frac{1}{N_y(1+G_s)}\right)^3 + \left(\frac{1}{N_y^{fd}}\right)^3\right]^{-\frac{1}{3}};$$
$$N_x^{fd} = \frac{9}{t_f^2}\left[D_{11} - \frac{D_{12}^2}{D_{22}}\right]\left(\frac{t_f}{t_d}\right)^{\frac{3}{2}}; \quad N_y^{fd} = \frac{9}{t_f^2}\left[D_{22} - \frac{D_{12}^2}{D_{11}}\right]\left(\frac{t_f}{t_d}\right)^{\frac{3}{2}}.$$

7. Optimization problem

The optimization problem can be formulated as finding the stacking sequences of the face sheet and the core thickness in order to minimize the weight W of the panel. The set of design variables is expressed as a vector $\tilde{t} = (\theta_1, \ldots, \theta_n, t_h)$, where n is an implicit design variable dictated by the number of layers in the face sheet stacking sequence. The optimization problem with displacement and buckling constraints can be expressed as

$$\min_{\tilde{t}} W(\tilde{t}), \tag{7}$$

such that

$$
\begin{array}{ll}
G_s(\tilde{t}) \geq 0 & \text{(strength constraint);} \\
G_b(\tilde{t}) \geq 0 & \text{(overall buckling);} \\
G_{fw}(\tilde{t}) \geq 0 & \text{(face wrinkling);} \\
G_{fd}(\tilde{t}) \geq 0 & \text{(face dimpling);} \\
t_h \in [t_h^l, t_h^u] & \text{(core thickness);} \\
\theta_i \in \{0^\circ, \pm 45^\circ, 90^\circ\}, \quad i = 1, \ldots, n & \text{(ply angles);}
\end{array}
$$

where $W(\tilde{t}) = [(2 \cdot t_f)\rho_f + t_h(8/3)(t_c/t_d)\rho_c]\,(a \cdot b)$ is the structural weight of the plate; ρ_c and ρ_f are material densities of the core and face ply, respectively; $t_f = \sum_{i=1}^{n/2} t_{ply}^i$ is the total face thickness; t_h^l and t_h^u are lower and upper bounds of the core thickness, respectively; θ_i is the orientation of the i-th ply; n is the total number of plies.

The critical constraint is defined as

$$
G_{cr}(\tilde{t}) = \min\{G_s, G_b, G_{fw}, G_{fd}\}, \tag{8}
$$

and the constrained optimization problem is transformed into an unconstrained maximization problem for the genetic algorithm. This is done by using penalty parameters. The fitness function Φ to be maximized is defined as

$$
\Phi(\tilde{t}) = \begin{cases} -W(\tilde{t}) + G_{cr}\delta, & G_{cr} \geq 0, \\ -W(\tilde{t})(1 - G_{cr})^p, & G_{cr} < 0, \end{cases} \tag{9}
$$

where δ and p are bonus and penalty parameters, respectively.

8. Results

A T300/N5208 graphite-epoxy unidirectional ply is used for the laminate face construction. The core material is HFT fiberglass reinforced phenolic honeycomb. The material properties of the face and the core are listed in Table 1.

Table 1. Material properties.

Face Place	Core
$E_1 = 20.5 \times 10^6$ psi.	$E_c = 23000$ psi.
$E_2 = 1.65 \times 10^6$ psi.	$G_{xz} = 19000$ psi.
$G_{12} = 0.75 \times 10^6$ psi.	$G_{yz} = 7500$ psi.
$\nu_{12} = 0.32$	$t_d = 0.125$ in.
$t_{ply} = 0.006$ in.	$t_c = 1.0$ in.
$\rho_f = 0.0055$ lbs/in ^{3}	$\rho_c = 0.00174$ lbs/in^3

The panel has the longitudinal and lateral dimensions of $a = 20$ in. and $b = 10$ in., respectively. The allowed range of the core thickness is from 0.1 in. to 1.5 in. The genetic algorithm was applied for one load case with $N_x = 8000.0$ lb/in and $N_y = 0.0$ lb/in. The GA stopping condition is either a limit on the total number of function evaluations conducted by the standard GA, or when there is no change in the fitness function between generations, whichever occurs first. For all the results here (other than Table 2, for which a limit of 125,000 was used as the stopping condition), the limit was set large enough that it was never reached.

8.1. Effect of binary tree

In order to demonstrate the efficiency that can be achieved through the use of a discrete memory, first consider the problem with the core thickness variable fixed to a constant value. The best known global optimal design for the specified load case obtained by the standard GA is presented in Table 2. The table gives the average number from ten runs of individuals ($\bar{n}_i$), the thickness (t), ply orientations (v), and fitness function value (Φ). This design was obtained with the constant core thickness, $t_h = 1.0$ in, in about 125000 function evaluations.

Table 2. Optimal design.

$\bar{n}_i$	t	v	Φ
125000	1.0	1111111	-1.3616

The efficiency of the binary tree for the fitness function evaluation is shown in Table 3. This table shows the averages from ten runs for the number of individuals (total number of attempts to evaluate fitness function) (n_i), the number of nodes in the binary tree (n_e), the number of repeated designs (n_r), savings in terms of amount of fitness evaluations (ξ), ply orientations (v), and fitness function value (Φ). The chromosome length is 20. It is evident from this table that 40–50 percent of the analyses can be avoided by using the GA with memory. This reduction of the total number of analyses agrees well with the results from the previous studies of the GA with memory [6], [7].

Table 3. The efficiency of the binary tree.

$\bar{n}_i$	$\bar{n}_e$	$\bar{n}_r$	$\bar{\xi}$, (%)	v	Φ
84572	43937	40635	47.0	1111111	-1.3616

8.2. Effect of spline interpolation

The results of this experiment were obtained with the core thickness varying in the range $[0.1, 1.5]$ in. The best known optimal design is presented in Table 4. The design was obtained using a standard GA with a large number of iterations requiring over a million function evaluations.

Table 4. Optimal design.

$\bar{n}_i$	t	v	Φ
1181040	0.4220	1111111	-1.1882

The performance of the GA with the spline interpolation is presented in Table 5, which shows averages from ten runs. Table 5 shows the best design after n_i attempts to evaluate the fitness function, n_s accepted spline values, and with $\epsilon = 0.005$ and $d_0 = 0.01$. The chromosome length is 7. The best known discrete vector v is $\{1111111\}$. The maximum absolute error δ_{max} and the mean absolute error δ_{av} are defined as

$$\delta_{max} = \max_{1 \le i \le n_s} \left| \Phi(x_i) - S(x_i) \right|, \qquad \delta_{av} = \frac{1}{n_s} \sum_{i=1}^{n_s} \left| \Phi(x_i) - S(x_i) \right|, \tag{10}$$

where n_s is the total number of acceptable spline evaluations. These errors are computed every time that the algorithm decides to extract an approximation of the fitness value without an exact analysis. The data show the cost of the GA with a continuous variable could be reduced up to 70%

Table 5. The efficiency of the spline interpolation.

$\bar{n}_i$	$\bar{n}_s$	$\bar{\xi}$, (%)	t	Φ	$\bar{\delta}_{max}$	$\bar{\delta}_{av}$
969011	896487	71.3	0.4220	-1.1882	0.11978	0.00006

relative to the standard GA by using the spline local approximation procedure. For the problem considered, the computation of the fitness function is not very expensive in terms of CPU time. However, this procedure has great potential in problems with expensive objective functions.

Table 6 shows he maximum absolute error δ_{max} and the mean absolute error δ_{av} for different values of the parameters ϵ and d_0. Table 7 shows the savings (ξ) in terms of number of fitness evaluations for different values of the parameters ϵ and d_0.

Table 6.
The errors of the spline interpolation (δ_{max}, δ_{av}) as a function of the parameters ϵ and d_0.

	d_0		
ϵ	0.005	0.01	0.02
0.005	0.03434, 0.00003	0.11978, 0.00006	0.20111, 0.00022
0.01	0.06142, 0.00004	0.09826, 0.00009	0.23898, 0.00024
0.02	0.04859, 0.00005	0.09676, 0.00012	0.32709, 0.00030
0.04	0.05555, 0.00012	0.10540, 0.00024	0.34188, 0.00042

Table 7. The percent savings (ξ) as a function of the parameters ϵ and d_0.

	d_0		
ϵ	0.005	0.01	0.02
0.005	50.5	70.8	75.7
0.01	59.6	72.1	78.5
0.02	63.0	76.2	80.8
0.04	64.9	77.8	86.4

Table 6 shows the expected trends; δ_{av} decreases as either ϵ or d_0 decreases, but δ_{max} is essentially random, being determined early in the iterations. Note that the average approximation error δ_{av} is quite acceptable, several orders of magnitude less than ϵ and d_0. Table 7 shows that the saving are significant, but of course these percentages will be smaller for larger chromosomes.

8.3. Effect of local improvement

The fitness as a function of the natural logarithm of the number of exact fitness evaluations is plotted in Figure 5. It can be observed that the algorithm with local improvement converges faster in terms of number of fitness function evaluations than the one with spline approximation. The algorithm with only spline interpolation demonstrates good convergence in comparison with the standard GA, and noticeably decreases the number of exact analyses. After about 20000 exact function evaluations the fitness values of the standard GA, the GA with spline interpolation, and the GA with local improvement are $\Phi = -1.2095$, $\Phi = -1.2031$, and $\Phi = -1.1882$, respectively.

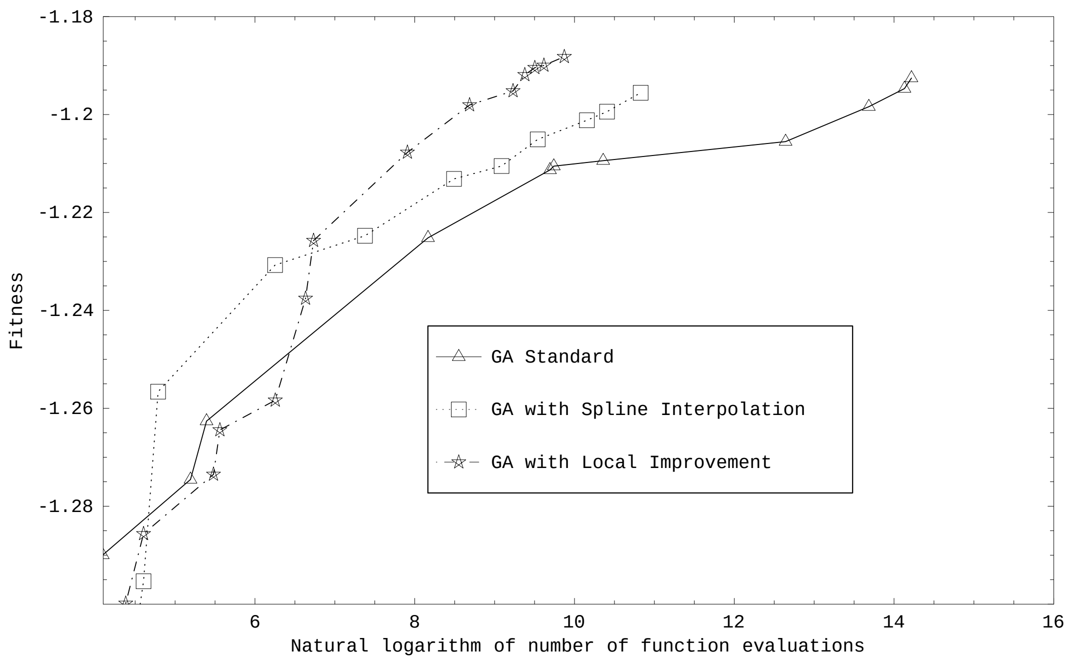

Fig. 5. Comparison of the performance (fitness vs. number of exact analyses).

9. Conclusions

A GA with memory along with spline interpolation was applied to the problem of weight minimization of a laminated sandwich composite panel with mixed discrete face sheet design variables and a continuous thickness design variable. The use of memory based on binary tree for discrete part of the design variables, and spline approximation for a continuous variable avoids repeating analyses of previously encountered designs. Moreover, it is also demonstrated that the spline approximation can be used to provide local improvement during the search and further reduce the number of exact function evaluations required to reach an improved solution. For FEM based analyses/simulations in which the computational cost is not trivial, the percent savings demonstrated in Tables 3, 5, and 7 will result in substantial computational savings. Even if less sophisticated analysis than FEM is used, the use of the binary tree will save time for all but extremely trivial function evaluations. Future work on this procedure might include development of a robust algorithm that is capable of handling several continuous variables using multivariate approximation techniques.

10. Acknowledgements

The authors are indebted to Christine Anderson-Cook for suggesting improvements in the approximation algorithm. 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] Schmit LA, Farshi B. Optimum design of laminated fiber composite plates. International Journal of Numerical Methods in Engineering 1977; 11: 623–640.

[2] Haftka RT, and Walsh JL. Stacking sequence optimization for buckling of laminated plates by integer programming. AIAA J. 1992; 3(3): 814–819.

[3] Nagendra S, Haftka RT, and Gürdal Z. Stacking sequence optimization of simply supported laminates with stability and strain constraints. AIAA J. 1992; 30(8): 2132–2137.

[4] Le Riche R, Haftka RT. Optimization of laminate stacking sequence for buckling load maximization by genetic algorithm. AIAA J. 1993; 31(5): 951–956.
[5] Seront G, Bersini H. A new GA-local search hybrid for optimization based on multilevel single linkage clustering. In: Proc. Genetic and Evolutionary Computation Conference (GECCO-2000), Las Vegas, Nevada, 10–12 July 2000.
[6] Kogiso N, Watson LT, Gürdal Z, Haftka RT. Genetic algorithms with local improvement for composite laminate design. Structural Optimization 1994; 7(4): 207–218.
[7] Kogiso N, Watson LT, Gürdal Z, Haftka RT, Nagendra S. Design of composite laminates by a genetic algorithm with memory. Mechanics of Composite Materials and Structures 1994; 1(1): 95–117.
[8] McMahon MT, Watson LT, Soremekun GA, Gürdal Z, and Haftka RT. A Fortran 90 genetic algorithm module for composite laminate structure design. Engineering with Computers 1998; 14: 260–273.
[9] Vowels RA. Algorithms and Data Structures in F and Fortran. Tucson, Arizona: Unicomp, Inc., 1998.
[10] de Boor C. A Practical Guide to Splines. Berlin: Springer-Verlag, 1978.
[11] Moh J, Hwu C. Optimization for buckling of composite sandwich plates. AIAA J. 1997; 35(5): 863–868.

List of figure captions.