ELSEVIER

Comput. Methods Appl. Mech. Engrg. 190 (2000) 1629–1650

Computer methods in applied mechanics and engineering

www.elsevier.com/locate/cma

Applying genetic algorithms to selected topics commonly encountered in engineering practice

K. Matouš, M. Lepš, J. Zeman, M. Šejnoha *

Department of Structural Mechanics, Faculty of Civil Engineering, Czech Technical University, Thákurova 7, 166 29 Prague 6, Czech Republic

Received 15 October 1999

Abstract

A carefully selected group of optimization problems is addressed to advocate application of genetic algorithms in various engineering optimization domains. Each topic introduced in the present paper serves as a representative of a larger class of interesting problems that arise frequently in many applications such as design tasks, functional optimization associated with various variational formulations, or a number of problems linked to image evaluation. No particular preferences are given to any version of genetic algorithms, but rather lessons learnt up-to-date are effectively combined to show the power of the genetic algorithm in effective search for the desired solution over a broad class of optimization problems discussed herein.

Keywords: Optimization; Genetic algorithm; Reinforced beam; Energy functional; Unit cell

1. Introduction

The search for a better, or rather an 'optimal' solution to a number of tasks that man has to face throughout his life is perhaps as old as mankind itself. Such peculiar competition aiming to outperform the others is quite similar to what nature does best through features like reproduction and self-adapting, so natural for each species striving for survival in a continuously changing environment. When accepting this common link we should not marvel at the immense problem-solving systems based on principles of evolution, that help to guide human decisions towards the 'best' alternative in a space of potential solutions.

A variety of *evolution programs* have been developed in the last few decades. For thorough discussion on this subject, we refer the interested reader to [3,4,9]. Here, we do not attempt to provide any major breakthrough in evolutionary programming but rather effectively exploit up-to-date knowledge to attain the goal we set. The main objective of this contribution is to manifest the robustness of evolution programs when applied to manifold optimization problems in engineering. We show that even a simple genetic algorithm can find its applicability in the design of engineering structures, where traditional gradient schemes become unacceptably expensive as they require an execution for a large number of different points to increase the chance in locating the global optimum, or cannot be used at all due to discontinuous nature of the objective function.

In keeping with the title of this paper, we address three specific problem areas of optimization. Section 2 outlines application of genetic algorithms (GAs) to a class of optimization problems associated with various

* Corresponding author. Tel.: +420-2-2435-4475; fax: +420-2-2431-0775.
E-mail address: sejnom@power1.fsv.cvut.cz (M. Šejnoha).

0045-7825/00/$ - see front matter
PII: S0045-7825(00)00192-4

design tasks. In particular, we attempt to minimize a cost of a steel-reinforced concrete beam. We search for a configuration characterized by a minimum price, which yet satisfies all strength and serviceability requirements for a given level of the applied load.

An application to a world of quadratic objective functionals polluted by various constraints appears in Section 3. The problem of interest represents a class of optimization tasks targeted at improving the overall performance of composite structures. As an example, we investigate the effect of the initial fiber pre-stress on both the local and overall response of the laminated plate. We wish to find an optimal distribution of the initial fiber pre-stress through the laminate thickness, which substantially increases the load-bearing capacity of laminates as well as reduces their maximum deflection. While the above problems are solved with the help of a binary version of genetic algorithms, Section 4 provides an example of floating-point representation of searched parameters.

The topic introduced in Section 4 falls into a category of image analyses. From the practical standpoint we address a community of researches interested in the field of random composites. Our objective is to generate a certain material representative volume element in terms of a periodic unit cell, which possesses similar statistical properties as the real composite. In this particular example the solution systems based on evolution strategies find its full potential, since the objective function is far from clean but rather noisy and discontinuous with a number of local valleys.

We realize that a reader not familiar with evolutionary programming might find himself trapped in meaningless pseudo-codes without properly rephrasing the vocabulary and essential features of genetic algorithms. However, instead of providing a general description of genetic algorithms in separate sections, we shall expose their peculiarities with a direct link to search parameters when studying individual problems.

2. Design of steel-reinforced concrete beam

A wide range of concrete materials in structural engineering in recent decades has led to many different optimization problems improving the design and overall performance of concrete structures. In most applications the aim has been at finding an optimum weight of a structure for given design conditions. To further enhance our problem, we add the total price of a structure into the gambling pool. Therefore, a standard task of designing structures for their maximum strength/weight ratio becomes a part of a more general picture.

To introduce the subject, consider a steel-reinforced concrete beam. The steel is usually characterized by its high strength and ductility, while concrete marks out by an advantageous pressure/strength ratio and price. When combining these two materials in a proper way a comparatively inexpensive structure can be obtained. Thus, we are after the less-expensive configuration that yet satisfies all strength and serviceability requirements.

When carefully examining this problem it becomes evident that an efficient and robust algorithm capable of handling a number of variable functions with discontinuities and nonlinearities is required. To tackle such a problem we may now rely on various stochastic algorithms with a genetic algorithm occupying an important place among them.

2.1. Objective function

Before proceeding with the actual description of a simple genetic algorithm and its modifications, we first formulate the desired objective function including penalty terms for incorporating various constraints.

One of the key quantities each design engineer takes into consideration is the price of a structure. Since an effective design of a structure can substantially reduce this quantity, we selected price as the objective function

$$f(\boldsymbol{X}) = V_c P_c + W_s P_s \tag{1}$$

subjected to following constraints:

$$\delta_i \leqslant \delta_{\lim} = \frac{l}{250}, \tag{2}$$

$$M_{Sd} \leqslant M_{Rd}. \tag{3}$$

In Eq. (1) V_c is the volume of concrete and W_s is the weight of steel; P_c and P_s are the price of concrete per unit volume and steel per kilogram, respectively. Inequalities (2) and (3) express selected design criteria according to EURODODE 2 standard (EC2) [12] for reinforced concrete (RC) structures. In particular, inequality (2) represents the serviceability requirement, where l is the span and $\delta_{\lim}$ is the maximum permissible deflection of a beam. Condition for the cross-section bearing capacity, inequality (3), given in terms of moments deserves more attention.

Consider a representative cross-section of reinforced concrete beam shown in Fig. 1 with given dimensions b and h. Internal forces acting on the cross-section, which are necessary for the design, are usually obtained using the finite element method.

According to [12] the required steel area is provided by

$$A_S = bd\frac{\alpha f_{cd}}{f_{yd}}\left(1 - \sqrt{1 - \frac{2M_{Sd}}{bd^2\alpha f_{cd}}}\right), \tag{4}$$

where M_{Sd} is the moment of internal forces. The ultimate moment M_{Rd} is then given by

$$M_{Rd} = A_S f_{yd}(d - 0.416x), \quad x = \frac{A_S f_{yd}}{0.81b\alpha f_{cd}}. \tag{5}$$

To handle inequalities (2) and (3) one may adopt a standard approach based on the penalty method. In such a case the original objective function (1) is augmented by including penalties for all constraints violations

$$f(\boldsymbol{X}) = V_c P_c + W_s P_s + \sum_{i=1}^{2} pf_i. \tag{6}$$

In our present approach the penalty functions pf_i assume, in general, the following form:

$$F_i \leqslant F_{i,\max}, \quad pf_i = \left(\frac{|F_i|}{F_{i\max} + A}\right)^B, \tag{7}$$

where A and B are the user-defined parameters of the proposed penalty function. Usually a large number is assigned to the parameter B, whereas A approaches zero.

At this point, however, we should warn the reader against perceiving the above approach as a general one for solving a constrained media problem. Although for a moderate number of constraints the

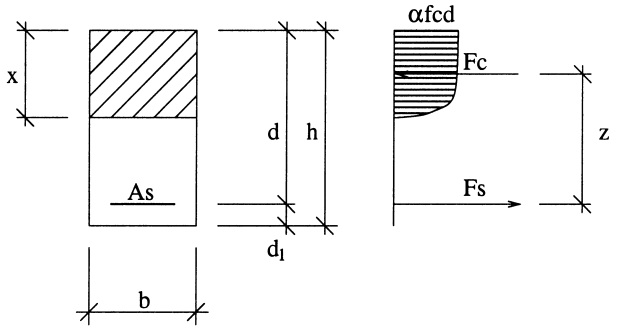

Fig. 1. RC beam with lower reinforcement.

approach, which includes penalties in the function evaluation, may prove to be reliable and efficient, the same might not be true when the number of constraints increases. In case of a larger number of constraints (shear strength requirements, and various other design criteria recommended by standards) it appears reasonable to follow, for example, an approach outlined in [9].

2.2. Optimization techniques

When designing an evolutionary program, one has first to square up to the principle question: Binary or float-point representation of searched variables. To shed light on this subject, we point out that most of the search variables in the above problem are either directly represented by integer numbers or as pointers to components of a discrete set of real numbers. Consequently, a binary alphabet appears as a natural choice for our representation space. Mapping between the representation and search spaces is described in the next section.

2.2.1. Data coding

To clearly understand the binary coding devised for this problem we first introduce the data structure for individual design parameters. From the genetic algorithm (GA) point of view, the only real-valued design parameters are the cross-sectional dimensions of a beam displayed in Fig. 2, where h and b represent the height and width of the cross-section, respectively. Table 1 lists the upper and lower bounds for each dimension together with the desirable precision. When implementing the GA we further assume that each dimension can either acquire discrete values spread 0.025 m apart or it can change continuously.

To introduce additional design variables recall Eq. (1) suggesting that steel reinforcement should be considered as important as concrete when attempting to reduce the cost of a structure. Table 2 stores the remaining 10 parameters selected to control an amount and location of the bending steel reinforcement. Note that all variables are treated as integer numbers.

In Table 2, pr_u and pr_b represent upper and lower reinforcement along the whole span of a beam, see Fig. 3. Parameters n_{bI}–n_{bIII} then correspond to the number of steel reinforcement bars located at the bottom of the beam cross-section within individual sections and n_{uI}–n_{uIII} stand for the number of bars located at the top of a beam, Fig. 3. For the design purpose, the beam is subdivided into a certain number of elements, where the internal forces are presumed to be constant. Parameters l_I and l_{II} are then associated with the

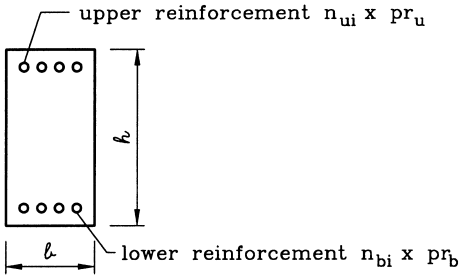

Fig. 2. Beam cross-section.

Table 1
Dimensions of the cross-section

Variable	Units	Minimum	Maximum	Precision	Comment
h	(m)	0.15	0.85	0.025	Discrete values
		0.15	0.85	0.001	Continuous values
b	(m)	0.15	0.45	0.025	Discrete values
		0.15	0.45	0.001	Continuous values

Table 2
Parameters of steel reinforcement

Variable	Minimum	Maximum	Precision	Comment
pr_u, pr_b	1	16	1	Indexes of vector of real numbers
n_{bI}–n_{bIII}	0	31	1	
n_{uI}–n_{uIII}	0	31	1	
l_I, l_{II}	0	127	1	No. of elements in a given interval

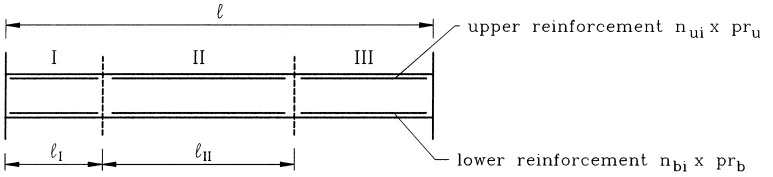

Fig. 3. Beam sections.

number of elements derived for sections I and II. l_{III} follows from a simple algebra. The above parameters can be stored in vector $\boldsymbol{X}$, Eq. (1), which in our particular case represents a vector of 12 variables.

We now proceed to construct a general mapping between the representation space and the search space, common for both integer and real numbers. In general, we consider a function $f(\boldsymbol{X}) = f(x_1, x_2, \ldots, x_n)$, where $\boldsymbol{X}$ is a vector of n variables, integer or real numbers x_i, defined on a closed interval

$$\min_i \leqslant x_i \leqslant \max_i, \tag{8}$$

where $\min_i$ and $\max_i$ are bounds assigned to each variable x_i taken from a domain D_i of either real or integer numbers. Further assume that each variable x_i is represented with some required precision p_i, defined as the smallest unit the number x_i can attain. Suppose that five decimal points for the real variable's precision are desirable, then $p_i = 0.00001$. If $D_i \subseteq N_0$ (integer numbers including zero) then $p_i = 1$. Provided that $p_i = 2$, the integer number x_i acquires either even or odd number depending on a given minimum, see Eq. (10). Each variable x_i can be transformed into a non-negative integer number $y_i \in N_0$ as

$$y_i = \left[\frac{x_i - \min_i}{p_i}\right]. \tag{9}$$

An inverse transformation is given by

$$x_i = y_i\, p_i + \min_i. \tag{10}$$

Ultimately, the number y_i is represented as a binary string of length k such that

$$\frac{\max_i - \min_i}{p_i} \leqslant 2^k. \tag{11}$$

An integer number k is provided by

$$k = \left[\frac{\ln\left(\frac{\max_i - \min_i}{p_i}\right)}{\ln 2}\right], \tag{12}$$

where operator $[z]$ denotes the integer part of z. It can be easily recognized that the length of a binary string depends quite substantially on the required precision. For example, coding a high-precision real number may lead to binary strings of size which essentially prevents the GA from successful implementation. In our study, however, such a weakness of binary GAs creates no obstacles.

Note that a binary form of vector $\boldsymbol{X}$ is often referred to as a *chromosome*, while individual variables x_i are termed *genes*. Thus, if our vector $\boldsymbol{X}$ was composed of two variables x, y, each represented by 8-bit binary number, then our *chromosome* would store two *genes* and consist of 16 binary digits. How chromosomes enter the GA procedure is discussed in the next section.

2.3. Genetic algorithms

Genetic algorithms, as stated in Section 1, are formulated using a direct analogy with evolution processes observed in nature, a source of fundamental difference between traditional optimizers and GAs. Genetic algorithms, in contrast to traditional methods, work simultaneously with a population of individuals, exploring a number of new areas in the search space in parallel, thus reducing the probability of being trapped in a local minimum. As in nature, individuals in a population compete with each other for surviving so that fitter individuals tend to progress into new generations, while the poor ones usually die out. This process is briefly described in Algorithm 1.

Algorithm 1. Principle of genetic algorithm

$t = 0$
generate P_0, evaluate P_0
while (**not** termination-condition) {
  $t = t + 1$
  select M_t from P_{t-1} (apply sampling mechanism)
  alter M_t (apply genetic operators)
  create P_t from M_t and evaluate P_t (insert new individuals into P_t)
}

Algorithm 1 provides basic steps of a single GA cycle; reproduction phase (#5), recombination (#6), and selection of a new population (#7). In the next paragraph we first explore basic operators controlling Step 6. Steps 5 and 7 will be explained in more detail when formulating various algorithms we tested.

2.3.1. Genetic operators

Breeding is the essential force-driving evolution of each species. Mating process, in which two parents combine their (we hope) good characteristics to produce (we hope) a better offspring, is accomplished in GAs through various 'cross-breeding' and 'mutating' operators. Detailed exposition to these operators is given in [4]. Here, we limit our attention to basic *crossover* and *mutation* operators we employed in the present study.

We begin with *uniform crossover*. When two individuals are selected for mating this operator works in accord with Fig. 4.

First, a random mask of the same length as the parents is generated. To create *offspring*-1 we proceed as follows. When the bit in the mask is 1 then the corresponding bit from *parent*-1 is copied to *offspring*-1 and when there is a 0 in the mask then the corresponding bit in *parent*-2 is passed to *offspring*-1. To create *offspring*-2 we simply exchange the order of parents. In addition, when randomly generating sets of ones and zeros, as shown in Fig. 5, we may arrive at *single-point* and *two-point* crossover operators, respectively. In other words, when applying single-point crossover, for example, we first randomly select a crossover

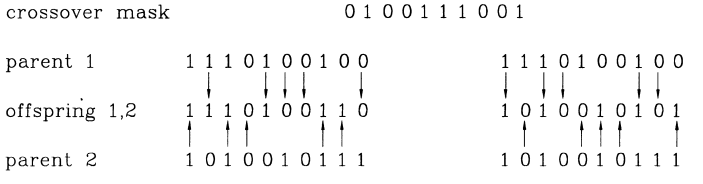

Fig. 4. Uniform crossover.

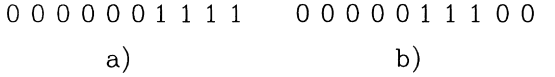

Fig. 5. (a) Single-point crossover mask; (b) Two-point crossover mask.

point after which parents exchange their tails. Crossover operator is usually applied with only a certain probability p_c. Therefore, not all pairs of individuals selected for mating are modified by the crossover step.

However, they still are bound to be disrupted by the *mutation* operator. The mutation operator randomly introduces new information, either positive or negative, into a population. It provides a very good job, particularly in the first exploration stage. However, its role in the recombination step should gradually decrease in the exploitation stage as the solution converges to the 'global' minimum. On the contrary, when a population gets trapped, over a certain number of GAs cycles, in a local minimum, the mutation operator might be the only source of a new information to drag the solution uphill to continue the search for the global minimum. The GA algorithm should be able to adaptively react to these contradictory effects. On the other hand, there exists several other, perhaps more appealing approaches which deal with this so-called 'premature convergence' towards a local minimum. Some of them will be discussed in Section 4.

In general, the mutation operator is applied to each new offspring created in the crossover step. For the binary algorithm, it just randomly changes bits from zeros to ones or vice-versa with a small probability, Fig. 6.

It is generally accepted that mutation plays a secondary role in a process of recombination, and as in nature the likelihood of its appearance is usually much smaller ($p_m = 0.001$–0.01) than that of crossover ($p_c = 0.6$–1). In what follows these operators will be placed within the framework of two simple versions of Algorithm 1 examined in our study.

2.3.1.1. Genetic algorithm I (GAB I). To keep up our promise given in Section 1, we start with a simple genetic algorithm described in [4] with only a minor difference related to sampling mechanism. This algorithm can be placed into the category of preservative, generational and pure selection procedures. It assumes non-zero selection probabilities for each individual. It carries out generational population replacement forcing each parent to reproduce in one generation only. To put this algorithm within the context of Algorithm 1 we now review the important steps in more detail:

Step 5. Individuals selected for reproduction are copied to the 'mating' pool according to their relative performance referred to as their 'fitness', or 'figure of merit'. In our case of function optimization, it is simply equal to the function value or rather its inverse when solving minimization problem. An expected number of copies each individual should receive in the mating pool M_t is given by

$$e_i = \frac{s_i}{\sum_1^N s_i} N, \quad s_i = \frac{1}{\delta + f_i}, \quad f_i \geqslant 0,$$

where N is the number of individuals in a population and f_i is the function value associated with the ith individual; $s_i = f_i$ when solving maximization problem. Parameter δ is a small positive number. To select individuals for recombination phase, we implemented a commonly used sampling mechanism called *remainder stochastic sampling without replacement* (RSSwoR) [2,4,9]. This method allocates individuals according to their integer part of e_i. The remaining places in a population are then sampled according to their fractional part using a spinning roulette wheel. The fractional parts represent a success probability of selection. After each spin, the expected value of the selected individuals is set equal to zero. As shown by Baker [2], however, this method is biased by favoring smaller fractions. Improved versions of this method provided by Baker are discussed in Section 4.

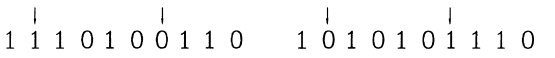

Fig. 6. Mutation.

Usually it is not desirable to sample individuals according to their raw fitness. In such a case the best individuals may receive a large number of copies in a single generation, so after a small number of GA cycles all individuals start to look alike and the algorithm usually converges prematurely to a local minimum. In other words, increasing the selection pressure decreases the population diversity. To compress the range of fitnesses, we incorporated a linear scaling (shifting) of the fitness function into our sampling procedure. For more details see, for example, [4]. However, care must be taken to avoid overcompression, which not only slows down the GA performance but may result in the loss of the global minimum [3].

Steps 6 *and* 7. Randomly select pairs of individuals from the mating pool M_t and perform recombination using genetic operators described in the previous paragraphs. Make sure that each individual is used only once. Replace individuals in P_{t-1} by a new offspring to create a new generation P_t.

2.3.1.2. Genetic algorithm II (GAB II). A number of GA confessors favor so-called *steady-state algorithms*, when only a few members of population are changed. A simple version of this approach is again outlined through individual steps of Algorithm 1.

Step 5. Reproduction phase employs the most simple sampling mechanism called *stochastic sampling with replacement* or simply the *Roulette wheel selection*. Details regarding its implementation are given in [4, Chapter 3]. In particular, by spinning the roulette wheel select two individuals from population P_{t-1} for mating. These individuals are temporarily stored in the mating pool M_t.

Step 6. Alter M_t by applying both the crossover and mutation operators, each with a prescribed probability.

Step 7. Using the inverse roulette wheel select two individuals from P_{t-1} marked to die out. Insert new offspring in P_{t-1} only if their relative performance is better than those selected for dying. Otherwise, there are no changes introduced in population P_{t-1}.

2.4. Examples

As an example we selected a continuous beam subjected to a uniformly distributed load according to Fig. 7. Due to symmetry, only one half of the beam was analysed. Distribution of internal forces (bending moment and shear force) was found using the finite element method. The required amount of steel then follows from Eq. (4). The price $P_c = 1350.0$ Kč/m^3 for concrete and $P_s = 50.0$ Kč/kg for steel was assumed.

To test the applicability of both algorithms, we explored two example problems. In the first example we attempted to reduce the price of a construction by merely modifying its shape. The steel remained unaffected. The second example dealt with the shape and bending-reinforcement optimization simultaneously.

In each case, an initial population of 200 individuals was randomly generated. Probabilities of crossover $p_c = 1$ and mutation $p_m = 0.03$ were kept constant throughout the GA run. Optimization process was terminated when there was no change in the best individual fitness observed over a certain number of GA cycles. Results appear in Tables 3–5.

Table 3 lists optimal dimensions of the beam cross-section together with corresponding price assuming both continuous and discrete change of cross-sectional dimensions during the optimization run. Both algorithms managed to find the exact minimum displayed in Fig. 8 (hollow circle). Function $f(b,h)$ in Fig. 8 is normalized with respect to a given price f_0. Here, f_0 represents the price derived from EC2 ($f_0 = 1002.65$ Kč, $b = 200,\ h = 300$ mm).

Results derived for the second example are stored in Table 4 showing the minimum, maximum, and average price associated with the best chromosome in a population. Standard deviation is added to

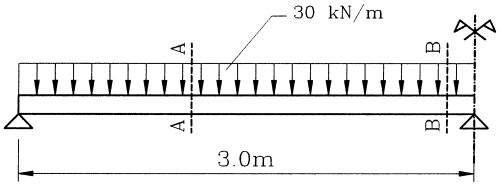

Fig. 7. Continuous beam subjected to uniform loading.

Table 3
Example 1

Values	b (mm)	h (mm)	Price (Kč)
Continuous	150	294	898.48
Discrete	150	300	906.39

Table 4
Example 2 – characteristics of the best individual

Algorithm	Values	Price (Kč)			
		Min.	Avg.	Max.	Std. dev.
GAB I	Continuous	851.27	866.24	880.83	9.01
	Discrete	959.07	977.09	1024.12	17.10
GAB II	Continuous	837.22	844.39	859.15	7.89
	Discrete	824.43	829.48	834.22	3.23

Table 5
Example 2 – results from five independent runs using GABII

Values	Section		Dimensions		Price (Kč)
	A–A	*B–B*	b (mm)	h (mm)	
Continuous	6 ∅ 6.0	10 ∅ 6.5	151	385	837.22
	5 ∅ 6.5	10 ∅ 6.5	150	393	842.21
	7 ∅ 5.5	12 ∅ 6.0	150	388	838.16
	6 ∅ 6.0	11 ∅ 6.0	150	408	859.15
	6 ∅ 6.0	10 ∅ 6.5	150	388	838.91
Discrete	5 ∅ 7.0	10 ∅ 7.0	150	350	824.43
	6 ∅ 6.5	10 ∅ 7.0	150	350	830.30
	7 ∅ 6.0	12 ∅ 6.5	150	350	833.18
	6 ∅ 6.5	12 ∅ 6.5	150	350	834.22
	7 ∅ 6.0	10 ∅ 7.0	150	350	829.27

complete the basic chromosome statistics. It is evident from Table 4 that a proper arrangement of the bending reinforcement bars can provide further reduction in the overall price. An additional improvement might be expected when considering the shear reinforcement as a part of the optimization process. This is the subject of the current investigation.

Although we tested all algorithms on one example only, results in Table 4 further suggest superiority of *steady-state* genetic algorithm GAB II over more traditional genetic algorithm GAB I, particularly when allowing only a discrete change of cross-sectional dimensions in the course of optimization. An absence of convergence observed in this case is attributed to the problem of having the population average fitness close to the population best fitness. Regardless of sufficient diversity within the population, both average and the best individuals reproduce in such a case with a similar number of copies in the next generations, which essentially reduces an opportunity for additional improvement. To remedy this situation, we may introduce a new source of information through randomly generated individuals to refresh a portion of the current population. However, we did not experiment with this approach. Fig. 9 displays convergence characteristics of both algorithms. Finally, Table 5 summarizes results obtained from five independent runs using the steady-state genetic algorithm. It shows variation in both dimensions and

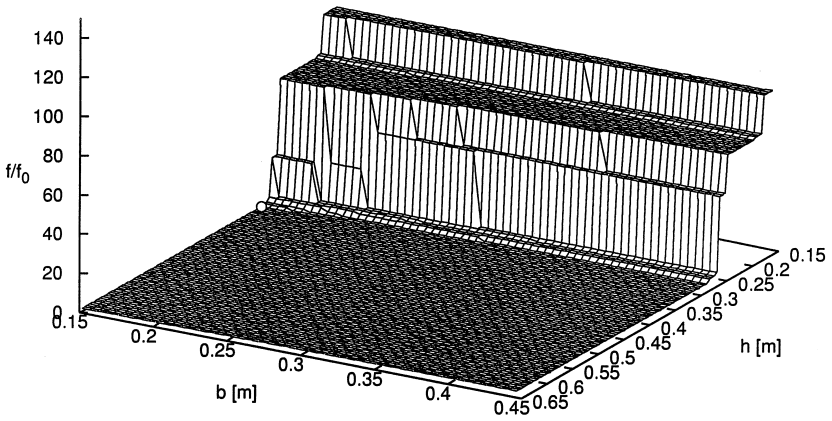

Fig. 8. Distribution of the objective function.

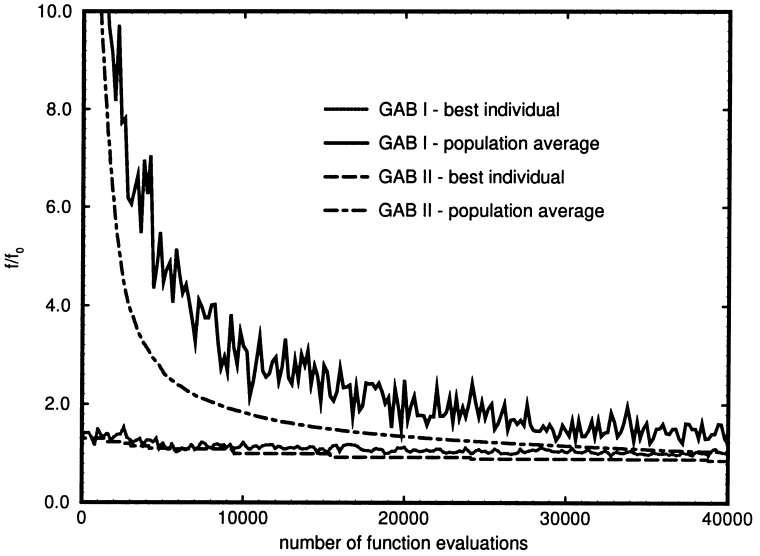

Fig. 9. Distribution of the objective function.

profiles of the lower (section A–A) and upper (section B–B) reinforcement bars, which yet manifests the discrete nature of the problem.

To conclude, we have shown ability of both algorithms to solve a simple design problem with only a few constraints. The above results, though encouraging when comparing the optimal price with the one from EC2, should not be overestimated as we avoided various design criteria recommended by standards. Complying with additional design requirements just increases a number of constrains, which eventually prevents the presented algorithms from working. Our early experiments, however, suggest that so-called *augmented simulated annealing*, also discussed later in this paper, is the right method of attack. Another possibility is to let genetic algorithms to do the hard work when exploring promising areas in the solution space initially and then call the local optimizer to descend individual hills in search for the best solution. These are the two routes we are currently pursuing.

3. Initially pre-stressed laminates

To stay on the moving train we examine in this section another problem from the category of structural design tasks. Although our objectives are essentially the same (improving the overall performance of

composite laminated structures), both the formulation of objective functional and our selection of the design parameters substantially differ from those introduced in the former section.

In this particular problem we combine a simple multilayered plate element with a micromechanical analysis of unidirectional fibrous composite plies to investigate the effect of the initial fiber pre-stress on both the local and overall response of the laminate. With the help of a certain energy-type objective functional, we seek for an optimal through-thickness variation of the initial fiber pre-stress in order to meet certain goals (reduce deflection, redistribute the local stresses in favor of the matrix phase, thus increasing the load bearing capacity of laminates). Here, the initial fiber pre-stress serves as a design parameter.

It is worthy to note that an efficient implementation of a genetic algorithm in its standard fashion suffers from a complicated and expensive cost functional, particularly when inelastic deformation of the matrix phase takes place during optimization. To overcome these obstacles we re-examine certain suggestions given in the literature [5] when dealing with such less-friendly objective functions or functionals. We show that even in the case when the local properties of the matrix phase together with its local stress and strain fields depend on the deformation history, the proposed optimization procedure built upon an augmented genetic algorithm still provides sufficiently accurate and efficient results.

3.1. Objective function

As in Section 2, we first turn our attention to the formulation of the objective functional. To this end, consider a laminated plate displayed in Fig. 10. The local displacement, stress and strain fields within individual plies are found using the refined laminated plate theory with separate assumptions for the displacement field of each layer [8]. The standard Mindlin kinematic conditions are assumed so that the displacement field takes the form

$$\begin{aligned} u_1^i(X_1,X_2,x_3) &= U_1^i(X_1,X_2) + x_3^i\phi_2^i(X_1,X_2),\\ u_2^i(X_1,X_2,x_3) &= U_2^i(X_1,X_2) - x_3^i\phi_1^i(X_1,X_2),\\ u_3^i(X_1,X_2,x_3) &= U_3^i(X_1,X_2), \quad i=1,2,\dots N, \end{aligned} \tag{13}$$

where x_3^i is measured from the middle plane of the ith ply; vector $\boldsymbol{U}^i$ represents axial displacements and ϕ_1^i, ϕ_2^i are rotations of the material line about the X_1, X_2 axis, respectively, within individual plies. Such a formulation is quite simple and straightforward, but it requires an introduction of the following constraints to maintain integrity of the laminate

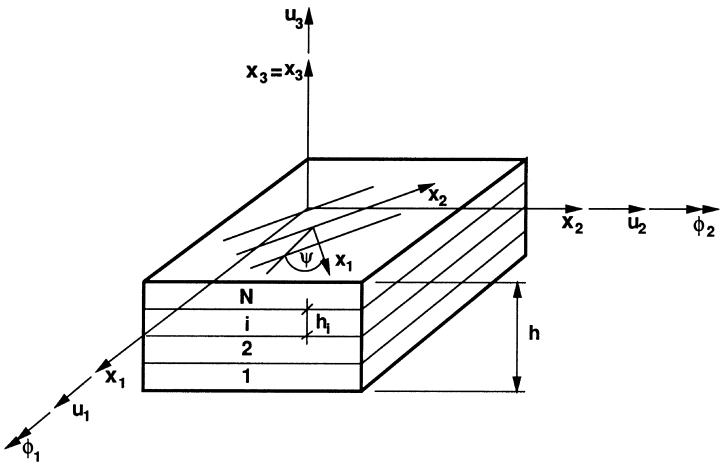

Fig. 10. Composite laminate.

$$
\begin{aligned}
g_1^i &= U_1^{i+1} - U_1^i - \frac{1}{2}\left[h^{i+1}\phi_2^{i+1} + h^i\phi_2^i\right], \\
g_2^i &= U_2^{i+1} - U_2^i + \frac{1}{2}\left[h^{i+1}\phi_1^{i+1} + h^i\phi_1^i\right], \quad i = 1, 2, \ldots N-1.
\end{aligned}
\tag{14}
$$

Actual implementation of this theory into a finite element code together with an incremental strategy for solving a nonlinear problem is not part of this study, and therefore we refer the reader to [14] for details on this subject. On the other hand, a short introductory invasion into the above layered laminated plate theory is necessary to understand individual terms in the proposed objective functional we now present in the following form:

$$
\Pi_0^j(\boldsymbol{u}, \boldsymbol{\beta}, \boldsymbol{\mu}) = U_m^j + \overline{U}_{\text{int}}^j + \int_{S_m} \sum_{i=1}^{\text{N}-1} (\boldsymbol{\beta}_i)^{\text{T}} \boldsymbol{g}_i \,\text{d}S + U_{\text{ext}}^j + \sum_{i=1}^{N} f_i. \tag{15}
$$

A graphical representation of the above expression is displayed in Fig. 11. In Eq. (15) the term U_{ext}^j represents the work done by externally applied load, vector $\boldsymbol{\beta}$ stands for the Lagrange multipliers for incorporating the displacement continuity conditions (14) and N is the number of layers of the laminated plate. The term $\overline{U}_{\text{int}}^j$ is the work of internal forces associated with bending and stretching effects. The last term in (15) provides certain constraints on fiber stresses to prevent their failure during the optimization process. Herein, it assumes the following form:

$$
f^i = \left(\frac{\sigma_{f_{11}}^i}{\sigma_u + A}\right)^B, \tag{16}
$$

where $\sigma_{f_{11}}^i$ is the actual tensile stress in the fiber and σ_u is the corresponding strength limit; A and B are again certain control variables defined by the user. The first quadratic term on the right-hand side of Eq. (15) U_m^j is given by

$$
U_m^j = \frac{1}{2}\int_{S_m} \sum_{i=1}^{N} \left[(\overline{\boldsymbol{\epsilon}} + \overline{\boldsymbol{\mu}})_i^{\text{T}} \mathscr{A}_i (\overline{\boldsymbol{\epsilon}} + \overline{\boldsymbol{\mu}})_i\right]^j \text{d}S_m, \tag{17}
$$

where $\mathscr{A}_i$ is the 3×3 plane stress tangent stiffness matrix multiplied by the ply thickness h_i. Contribution to the overall initial strain vector $\overline{\boldsymbol{\mu}}$ is caused by the uniform fiber pre-stress $\boldsymbol{\lambda}_{\text{f}}^{\text{pr}}$ ($\boldsymbol{\mu}_{\text{f}}^{\text{pr}} = -\boldsymbol{M}_{\text{f}}\boldsymbol{\lambda}_{\text{f}}^{\text{pr}}$; $\boldsymbol{M}_{\text{f}}$ is the compliance matrix of the fiber phase), and by the plastic strain developed in the matrix. Finally, $\overline{\boldsymbol{\epsilon}}$ stores components of the total in-plane strain vector.

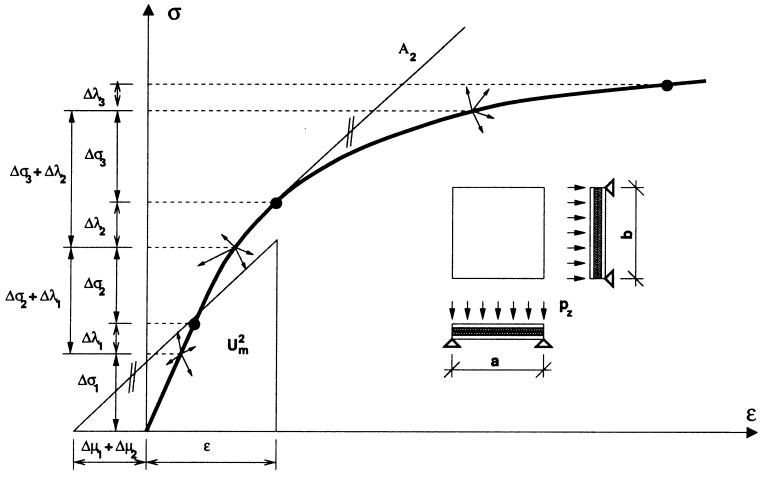

Fig. 11. Formulation of objective functional.

Note that Eq. (15) is written for individual load increments. This implies that the optimization procedure is called to generate an increment of the initial fiber pre-stress, which minimizes (15) for the current jth increment of the prescribed load. Evidently, Eq. (15) is not exactly a clean quadratic functional, but rather represents a constrained media problem. Moreover, its evaluation takes up most of the time spent on a single genetic algorithm cycle. How to reduce the number of its evaluation and yet maintain efficiency of the algorithm is discussed in the next section.

3.2. Optimization technique

In this section we continue our experimentation with binary coding and simple crossover. In view of several important remarks given in the preceding paragraph we work with a very small population consisting of two individuals only. In the search space, an individual is formed as a vector of n variables corresponding to the initial fiber pre-stress $(\boldsymbol{\lambda}_{\mathrm{f}}^{\mathrm{pr}})_i$ applied in individual plies ($n \leqslant N$). Each variable is then mapped to its counterpart in the representation space following the procedure described in Section 2.2.1. A binary representation of searched variables is displayed in Fig. 12.

To construct a single GA cycle we have to bear in mind a loading-path-dependent response of a structure undergoing an inelastic deformation. To understand this subject, assume that a structure with a certain deformation history arrives at the equilibrium state for a given level of external loading and magnitudes of the initial fiber pre-stress derived up to the last but one load increment. Our objective now is to adjust the level of the initial fiber pre-stress to minimize Eq. (15). Further assume that the best solution obtained so far is stored. In the representation space we call this solution chromosome 1. Only the best solution is stored and for a new reproduction cycle the second parent is generated randomly (chromosome 2).

To include reproduction step in the GA cycle we select parents to breed among individual genes as shown in Fig. 12 according to their fitness. To this end, a standard roulette wheel is used. Pairs of genes for breeding are drawn from the best chromosome and their number is arbitrary. To determine an influence of individual genes on the objective function we load the structure, in turn, by increments of the n components of the solution vector (increment of the initial fiber pre-stress resulting from genetic operations). Mutation operator can be either applied similarly as in the previous section or omitted entirely due to the fact that a new vector is generated after each cycle. In our study, the mutation operator was included but with a rather low probability $p_{\mathrm{m}} = 0.01$.

3.3. Example

As an example we present results for the $[0/90]_{\mathrm{s}}$ simply supported composite laminate loaded in transverse compression. Each ply is made of aligned T-50 graphite fibers bonded to the 6061-O aluminum matrix with $c_{\mathrm{f}} = 0.5$. The solution vector thus consists of four variables. The search space is defined on the closed interval $\langle -100, \ldots, 100 \rangle$ MPa.

Fig. 13 shows a distribution of the fiber pre-stress during optimization and loading process. Note that the optimization procedure is called continuously for each load increment throughout the entire loading program. The solid line corresponds to a fiber pre-stress generated in the most bottom ply. Since no constraints were imposed on fiber pre-stress signs, the method also produced a negative fiber pre-stress in

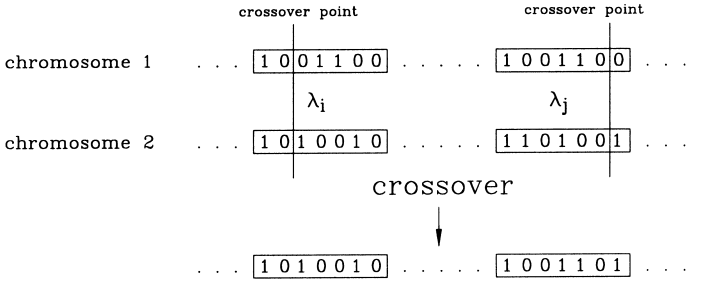

Fig. 12. Representation space and crossover operator.

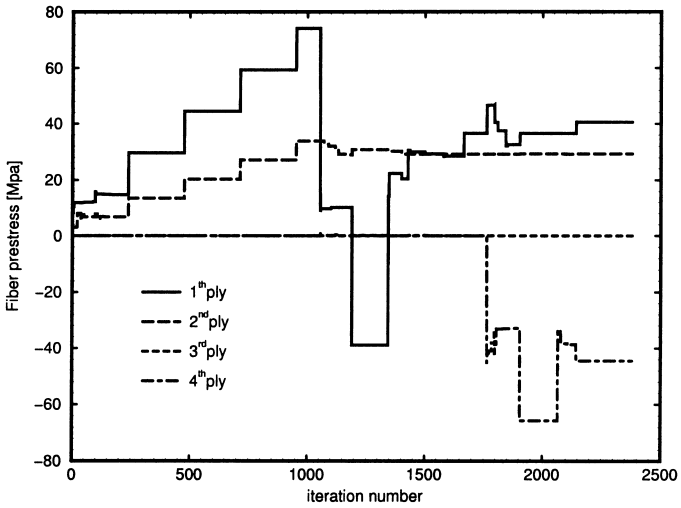

Fig. 13. Distribution of fiber pre-stress during optimization.

the most upper ply which is of the same magnitude as the one generated in the first ply to compensate for the bending moment due to the applied load. In reality, however, only a tensile fiber pre-stress can be introduced during the manufacturing process. This fact has to be taken into account when setting up the optimization process.

Nevertheless, the final redistribution of the initial fiber pre-stress not only substantially reduced the maximum vertical deflection of the laminate, but also led to a final redistribution of local stresses in favor of matrix, which provides an additional improvement in the laminate response. Such encouraging results indirectly support the proposed optimization procedure, and also proves applicability of GAs in problems with complicated functions.

4. Formulation of a periodic unit cell

The last section of this paper is devoted to a community of researchers interested in micromechanical analysis of composite materials. Here, we limit our attention to one specific problem associated with a formulation of the representative volume element (RVE) for a composite medium with randomly distributed fibers. To introduce a subject, imagine a high contrast micrograph of a part of the graphite fiber tow impregnated by the polymer matrix displayed in Fig. 14.

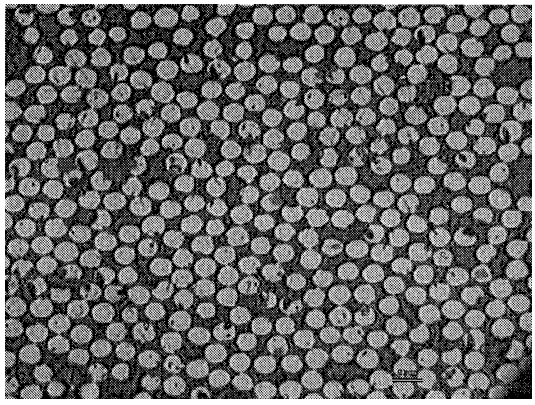

Fig. 14. A micrograph of the fiber tow.

This sample typically consists of a large number of particles (in order of 100) and its direct analysis would be rather expensive. To remedy this problem we follow a clever idea proposed by Povirk [11]. Instead of examining a large micrograph, Fig. 14, he suggested to formulate a representative volume element in terms of a periodic unit cell, which closely resembles the real microstructure. A number of ways can be used to accomplish this task. Here, we offer a simple approach based on microstructural statistics.

4.1. Objective function

The approach we are going to discuss in this section requires a knowledge of a certain microstructure-characterizing function. An example of such a function is the *second-order intensity function* $K(r)$ [13]. This function is defined as the number of further points expected to lie within a distance r of an arbitrary point divided by the number of points per unit area

$$K(r) = \frac{A}{N^2} \sum_{i=1}^{N} I_i(r), \tag{18}$$

where A is the area of a sample, N the number of points in a given sample and $I_i(r)$ is the number of further points within a circle with center at point i and radius r. For more details about this function see, e.g., [1]. Providing this function is available for the original microstructure, we argue that the RVE can be represented by a periodic unit cell with only a few reinforcements, for which this function is most similar to the original one. After accepting this assumption we may formulate the following problem:

For a given number of fibers N, dimensions of a unit cell H_1 and H_2 and values of the original function $\overline{K}(r)$ evaluated in points $r_i,\ i = 1, \ldots, N_m$ find the configuration of particle centers $\boldsymbol{x}^N_{H_1,H_2}$ such that

$$\boldsymbol{x}^N_{H_1,H_2} = \arg \min_{\boldsymbol{x} \in S} F(\boldsymbol{x}^N_{H_1,H_2}) \quad \text{where} \quad F(\boldsymbol{x}^N_{H_1,H_2}) = \sum_{i=1}^{N_m} \left(\frac{\overline{K}(r_i) - K(r_i)}{\pi r_i^2} \right)^2, \tag{19}$$

where vector $\boldsymbol{x} = \{x^1, y^1, \ldots, x^N, y^N\}$ stands for the configuration of particle centers of the periodic unit cell; x^i and y^i correspond to x and y coordinates of the ith particle and $\mathscr{S}$ denotes a set of admissible vectors $\boldsymbol{x}$. After solving the above problem for fixed values of the unit cell dimensions we may further adjust H_1 and H_2 to cover all reasonable values. The most suitable RVE then corresponds to the minimal attained value of $F(\boldsymbol{x}^N_{H_1,H_2})$.

Our choice of function $K(r)$ for the problem optimization is primarily attributed to its simplicity and very rapid evaluation for a reasonable number of particles N. In addition, the selected form of the objective function F, Eq. (18), is quite useful since it serves directly as a 'natural' penalization when particles happen to overlap. Therefore, no additional algorithmic labor necessary for avoiding unacceptable solutions [11] is needed.

To define a set $\mathscr{S}$ of admissible solutions we recall the principle objective: construction of a periodic unit cell. Such a unit cell is then surrounded by periodic replicas of itself so x^i and y^i can take arbitrary values. To avoid various numerical difficulties associated with the presence of two materials at the boundary, we further require that a set of vectors $\boldsymbol{x}$ in (19) satisfies the following condition:

$$R \leqslant x^i \leqslant H_1 - R \ \wedge \ R \leqslant y^i \leqslant H_2 - R, \quad i = 1, \ldots, N, \tag{20}$$

where R is the fiber radius. Eq. (19) together with condition (20) then represent a constrained optimization problem. Several tips for dealing with such a problem are given in [9]. To further support our selection of genetic algorithms for solving Eq. (19) we present an admissible unit cell consisting of 10 fibers together with an example of the objective function F manifesting the problem complexity, Fig. 15. Coordinates x^1 and y^1 in Fig. 15 represent locations of the filled fiber center. Positions of remaining fibers are fixed. The minimum of function F is marked by a hollow circle.

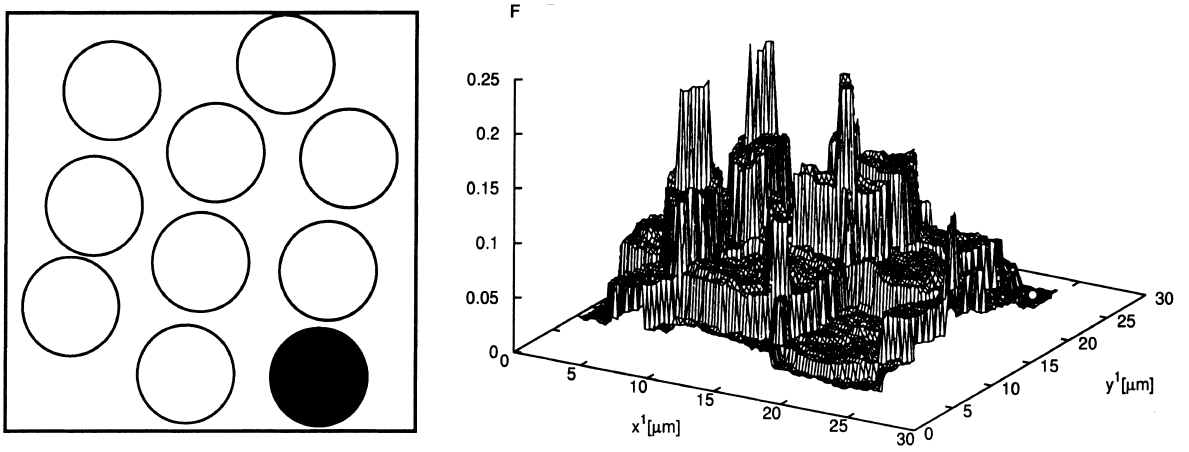

Fig. 15. An admissible unit cell.

4.2. Optimization techniques

In Sections 2 and 3, we discussed an application of the binary version of genetic algorithms. The basic ideas and vocabulary of genetic algorithms we have reviewed so far will also be adopted herein, although in a somewhat different manner.

In Section 2, we have already pointed out certain drawbacks associated with the binary coding of searched variables. In particular, we recall a precision limitation of the binary genetic algorithm by the binary representation of its parameters. This problem becomes particularly important when the search space is formed by high-precision continuous parameters, since a high-precision requirement usually implies a very large representation space. In such a case the binary genetic algorithm performs rather poorly.

In view of this general conclusion we now abandon the binary representation of parameters for the remaining part of this study and turn our attention to a floating-point representation of genes instead. This step brings a number of advantages. First of all, using real numbers easily allows representation to the machine precision. In addition, the search operators work directly in the search domain thus no mapping between the representation space and the search space is required. This is a direct consequence of the floating point implementation, where each chromosome vector is coded as a vector of floating point numbers, of the same length as the solution vector.

Various modification of the continuous genetic algorithm will now be described in the sequel. For additional comparison we also present an augmented version of the simulated annealing method, see [7].

To begin we recall Algorithm 1, which summarizes essential steps of the genetic algorithm. The population P_t now becomes a family of possible configurations of a single unit cell. Each individual, a unit cell, is represented by a real-valued vector $\boldsymbol{X} = \{x_1, \ldots, x_{2N}\}$. Individual components of this vector are related to actual fiber centers as follows:

$$x_{2i-1} = x^i \text{ and } x_{2i} = y^i \quad \text{for } i = 1, \ldots, N,$$

where N is the number of fibers within the unit cell. As for the binary algorithm, step 6 in Algorithm 1 requires application of certain genetic operators acting on individual chromosomes (vector $\boldsymbol{X}$). The next Subsection provides a list of operators developed for float-point genes, which were implemented in our codes. Details regarding their construction can be found in [10].

4.2.1. Genetic operators for real-valued alphabets

Let L_i and U_i represent the lower and upper bound for each variable x_i, respectively. Further assume that vector $\boldsymbol{X}$ represents a parent, whereas vector $\boldsymbol{X}'$ corresponds to an offspring; $u(a, b)$ is a real number and $u[a, b]$ is an integer number with uniform distribution defined on a closed interval $\langle a; b\rangle$. The following operators can now be defined:

Uniform mutation. Let $j = u[1, 2N]$ and set

$$x_i' = \begin{cases} u(L_i, U_i) & \text{if } i = j, \\ x_i & \text{otherwise.} \end{cases}$$

Boundary mutation. Let $j = u[1, 2N]$, $p = u(0, 1)$ and set

$$x_i' = \begin{cases} L_i & \text{if } i = j,\ p < 0.5, \\ U_i & \text{if } i = j,\ p \geqslant 0.5, \\ x_i & \text{otherwise.} \end{cases}$$

Non-uniform mutation. Let $j = u[1, 2N]$, $p = u(0, 1)$ and set

$$x_i' = \begin{cases} x_i + (L_i - x_i) f(t) & \text{if } i = j,\ p < 0.5, \\ x_i + (U_i - x_i) f(t) & \text{if } i = j,\ p \geqslant 0.5, \\ x_i & \text{otherwise,} \end{cases}$$

where $f(t) = u(0, 1)(1 - t/t_{\max})^b$, t is the current generation, $t_{\max}$ the maximum number of generations and b is the shape parameter. This operator allows for a local tuning as it uniformly searches the space initially and very locally at later stages.

Multi-non-uniform mutation. Non-uniform mutation applied to all variables of $\boldsymbol{X}$.

Simple crossover. Let $j = [1, 2N]$ and set

$$x_i' = \begin{cases} x_i & \text{if } i < j, \\ y_i & \text{otherwise,} \end{cases}$$

$$y_i' = \begin{cases} y_i & \text{if } i < j, \\ x_i & \text{otherwise.} \end{cases}$$

Simple arithmetic crossover. Let $j = u[1, 2N]$, $p = u(0, 1)$ and set

$$x_i' = \begin{cases} px_i + (1 - p)y_i & \text{if } i = j, \\ x_i & \text{otherwise,} \end{cases}$$

$$y_i' = \begin{cases} py_i + (1 - p)x_i & \text{if } i = j, \\ y_i & \text{otherwise.} \end{cases}$$

Whole arithmetic crossover. Simple arithmetic crossover applied to all variables of $\boldsymbol{X}$.

Heuristic crossover. Let $p = u(0, 1)$ and set

$$\boldsymbol{X}' = (1 + p)\boldsymbol{X} - p\boldsymbol{Y},$$

$$\boldsymbol{Y}' = \boldsymbol{X},$$

where $\boldsymbol{X}$ is a better individual than $\boldsymbol{Y}$ in terms of fitness. If $\boldsymbol{X}' \notin \mathscr{S}$, then a new random number p is generated until the feasibility condition ($\boldsymbol{X}' \in \mathscr{S}$) is met or the maximum allowed number of heuristic crossover applications is exceeded.

Altering an old generation P_{t-1} requires an application of each operator in a certain number of times depending on the selected sampling mechanism. Recall that the sampling mechanism facilitates the reproduction step of the GA (step 5 in Algorithm 1).

We already know from our work in Section 2 that a proper selection scheme may quite significantly influence an ultimate performance of the GA. However, there is no firm evidence proving the superiority of one particular sampling method over the others. This bare fact, on the other hand, reveals the real beauty of GAs as it provides a relative freedom when developing a new evolutionary program. The versatile phenomenon of GA-based optimization approach is further demonstrated in the next section, where we present some additional modifications of a simple genetic algorithm for enhancing the genetic search.

4.2.2. Genetic algorithms

This section continues our exposition to popular sampling mechanisms available in the literature. Each method is again placed within the context of particular genetic algorithm. We describe them in turn by referencing individual steps of Algorithm 1.

4.2.2.1. Genetic Algorithm I (GAR I). We begin with the most simple one usually termed as *steady-state GAs*. Reproduction is implemented through the weighted roulette wheel and only one or two offspring are created within each generation. For better understanding we now review the relevant steps:

Step 5. By spinning the roulette wheel, select the r individuals from population P_{t-1} required for mating (one individual for mutation, two individuals when the crossover operator is applied). These individuals are temporarily stored in the mating pool M_t. Linear scaling is used to reduce a common threat of premature convergence to a local optimum.

Step 6. Altering M_t by applying either crossover or mutation operator. In our case, the mutation operator is used twice as often as the crossover operator.

Step 7. Based on a number of new offsprings created, select a corresponding number of individuals from P_{t-1} to die using the inverse roulette wheel. Insert new offsprings into P_{t-1} to create P_t.

4.2.2.2. Genetic Algorithm II (GAR II). This algorithm closely resembles the simple genetic algorithm described in Section 2 with only minor changes. To reduce statistical errors associated with the roulette wheel selection we employ, in this case, an improved version of RSSwoR called *remainder stochastic independent sampling* (RSIS) [2].

Unlike the RSSwoR, where the fractional expected values are sampled spinning the roulette wheel, the RSIS uses each fractional value as a probability of selection. In particular, when sampling the fractional parts we first generate a random real number r uniformly distributed within the range $[0..1]$. Then we check, whether the fractional part of the ith individual is greater than r. If yes, the individual is added to the mating pool and its fractional part is set equal to zero. If not, we move to the next individual. These steps are repeated until the number of individuals in the mating pool equals the population size. This algorithm is substantially simpler than the RSSwoR and the number of operations needed for reproduction is of order of N (population size). As for GAR I, we now review the important steps of Algorithm 1:

Step 5. Apply the RSIS sample individuals from P_{t-1} copy them into the mating pool M_t. Note that precisely N individuals are selected for reproduction. This sampling method thus falls into the category of preservative and generational selections [9]. Similar actions as in GAR I are taken to deal with the premature convergence.

Step 6. Genetic operators are applied to *all* individuals in M_t. Each operator is used in a prescribed number of times depending on the population size, and new individuals are placed into a temporary population P'_t. Parents for breeding are selected uniformly.

Step 7. Create a new population P_t by successively replacing the worst individual from P_{t-1} by individuals from the temporary population P'_t.

4.2.2.3. Genetic Algorithm III (GAR III). This algorithm is essentially a replica of the Michalewicz modGA algorithm ([9, p. 59]). It employs the *stochastic universal sampling mechanism* (SUS) presented by Baker [2]. It is based on a single roulette wheel spin. In Baker's formulation, the standard roulette wheel is marked with equally spaced pointers indicating the happy individual selected for reproduction. A number of pointers indicates a number of desired individuals used for reproduction. This is particularly appreciable when applying the modGA, which is characterized by the following steps:

Step 5*a*. Using the SUS, select a subset of n individuals from P_{t-1} for reproduction and copy them to M_t. Note that each member of M_t can appear only once in the reproduction cycle.

Step 5*b*. Again using the SUS, select exactly $N-n$ individuals from P_{t-1} and copy them to a new population P_t.

Step 6. Select uniformly parents from M_t to produce exactly n offsprings (as in *GAR II*, but in this case the genetic operators act only on n individuals stored in the mating pool).

Step 7. Add new offspring to population P_t.

4.2.2.4. Hybrid genetic algorithm (HGA). GAs are generally very efficient in finding promising areas of the searched solution. On the other hand, they may perform rather poorly when shooting for the exact solution with a high degree of precision (premature convergence, convergence to local optimum, etc.). Therefore it appears logical to combine GAs exploring the search space initially with a suitable gradient or deterministic optimizer exploiting promising solutions locally.

As natural for GAs, this procedure can be implemented in a number of ways. When experimenting with this approach, we combined various ideas suggested up-to-date, which eventually led to a reliable and efficient algorithm. It works with relatively small population sizes, which makes computationally feasible to restart the genetic algorithm after a given convergence criterion is met. Each restart is associated with a certain number of new members entering the initial population to maintain a sufficient diversity among chromosomes. Consequently, mutation operators can be excluded from reproduction. Individual steps of this algorithm are now discussed in a sequel:

Step 2. Randomly generate a small population.

Steps 5 *and* 6. Perform standard genetic operations until convergence or the maximum number of generations exceeded. To select chromosomes for reproduction we applied *stochastic tournament selection* scheme [4,3]. Only crossover operators are used.

Step 7*a*. Select the n of best individuals for local search. We adopted the *dynamic hill climbing* method of Yuret [15] and his *local optimize* to seek for the desired optimum with a starting point provided by the GA. When the local optimizer converges copy new individuals into P_t.

Step 7*b*. Add $N - n$ randomly generated individuals to fill population P_t. This ensures diversity among chromosomes. Go to step 5 and restart the GA.

4.2.2.5. Augmented simulated annealing (AUSA). When talking about GAs, it would be unfair not to mention another popular method using random choice to explore the solution space, namely the *augmented simulated annealing method* presented by Kvasnicka [7]. This method effectively exploits the essentials of GAs (a population of chromosomes, rather than a single point in space, is optimized) together with the basic concept of simulated annealing method guiding the search towards minimal energy states.

If we wish to put GAs and the AUSA on the same footing, we may relate the AUSA to a group of *steady state* and *on the fly* methods [9], in which the offspring replaces its parents immediately. The replacement procedure is controlled by the Metropolis criterion, which allows a worse child to replace its better parent with only a certain probability. The probability of excepting a worse solution is reduced as the procedure converges to the 'global' minimum. The following algorithm describes an implementation of the AUSA:

Algorithm 2. Augmented simulated annealing

$T = T_{\max},\ t = 0$
generate P_0, evaluate P_0
while (**not** termination-condition) {
  $counter = success = 0$
  while($counter < counter_{max} \wedge success < success_{max}$) {
    $counter = counter + 1,\ t = t + 1$
    select operator O
    select individual(s) I_t from P_t
    modify I_t by O
    select individual(s) I'_t from P_t
    $p = \exp((F(I'_t) - F(I_t))/T)$
    if $(u(0,1) \leqslant p)$ {
      $success = success + 1$
      insert I_t into P_t
      evaluate P_t
    }
  }
  decrease T
}

Step 7. It is recommended to choose mutation operators with much higher probabilities then crossovers. In [7] ratio ≈ 0.1 is proposed.

Step 8. New individuals are selected using the *normalized geometric ranking* method [6].

Step 10. Instead of replacing parents we select individuals to die using the inverse normalized geometric ranking method.

Step 11. The temperature $T_{\max}$ should be chosen such that the ratio of accepted solutions to all solutions is $\approx 50\%$.

Step 18. This step is called the *cooling schedule*. We use a very simple form of cooling schedule $T_{i+1} = T_{\text{mult}} T_i$. In this step we also perform *reannealing* if necessary. If the actual temperature is lower than a given parameter $T_{\min}$, we set $T = T_{\max}$ and copy half of the current population to a new one. Remaining part of the new population is generated randomly.

4.2.2.6. Diversity of population. As is evident from Fig. 15, the objective function possesses quite a large number of plateaus. Thus a part of population inevitably lands on one of these plateaus if no action is taken. This, however, substantially decreases performance of the genetic algorithm.

To overcome this obstacle, we introduce a very simple procedure for maintaining a sufficient diversity in population: Before inserting an offspring $\boldsymbol{X}$ into population, we first search for an individual $\boldsymbol{X}'$ which satisfies

$$F(\boldsymbol{X}) = F(\boldsymbol{X}') \tag{21}$$

$$\max_i |x_i - x_i'| < \varepsilon, \quad i = 1, \ldots, 2N, \tag{22}$$

where ε is set here to 1×10^{-5}. If such an individual exists, it is replaced by $\boldsymbol{X}$. Otherwise an individual $\boldsymbol{X}$ enters a population following step 7 in above algorithms. This procedure, though very simple and 'naive', yields substantial improvement in stability above all previously mentioned methods.

4.3. Examples

To test individual methods, we assumed a square periodic unit cell consisting of 10 fibers with the same volume fraction as the real specimen. As a first step we wished to fit functions $\overline{K}(r)$ and $K(r)$ in five points only ($N_m = 5$). Sampled points were spaced by fiber diameter.

In all cases, the initial population was generated purely randomly. Except for the HGA we created a population of size equal to 64 chromosomes. Only eight individuals were generated to fill a population when running the HGA. Iteration process was terminated, if one of the following conditions was met:

- Algorithm returned value $F(\boldsymbol{x}) \leqslant \varepsilon = 6 \times 10^{-5}$.
- Number of function evaluations exceeded 250,000.

Each algorithm was run 20 times. For each run, the number of function evaluations was recorded together with the minimum attained value of the objective function (19).

Table 6 shows the minimum, maximum, and average values for the number of function evaluations. Table 7 lists similar results for the best chromosome in a population. In this case, however, we also included runs terminated after exceeding the maximum number of function evaluations. Presented results provide no evidence for promoting one particular method and discriminate the others, although nobody is perhaps caught by surprise seeing the HGA as the current winner and the GAR I, which did not always converged, as a loser. On the other hand, since all properties affecting the searching process (population size, initial parameter settings relevant to individual methods, age and optimization process dependent probabilities guiding an application of a given genetic operator) are hand-tuned only, the efficiency of algorithms can be underestimated.

To check quality of the resultant unit cell we plotted the second-order intensity function for original microstructure ($\overline{K}(r)$) against the one associated with a unit cell chosen as the minimum from 20 independent runs. Results, which appear in Fig. 16, were derived via the AUSA method. Evidently, both

Table 6
Number of function evaluations

Algorithm	Number of evaluations		
	Min.	Avg.	Max.
GAR I	8896	74,562	193,600
GAR II	6956	17,270	55,296
GAR III	4484	12,037	26,224
HGA	1613	8856	24,404
AUSA	3490	8709	26,314

Table 7
Characteristics of the best individual

Algorithm	Number found	Returned value $\times 10^5$		
		Min.	Avg.	Max.
GAR I	18	6.0	7.2	16.4
GAR II	20	5.9	6.0	6.0
GAR III	20	5.9	6.0	6.0
AUSA	20	5.9	6.0	6.0
HGA	20	5.9	6.0	6.0

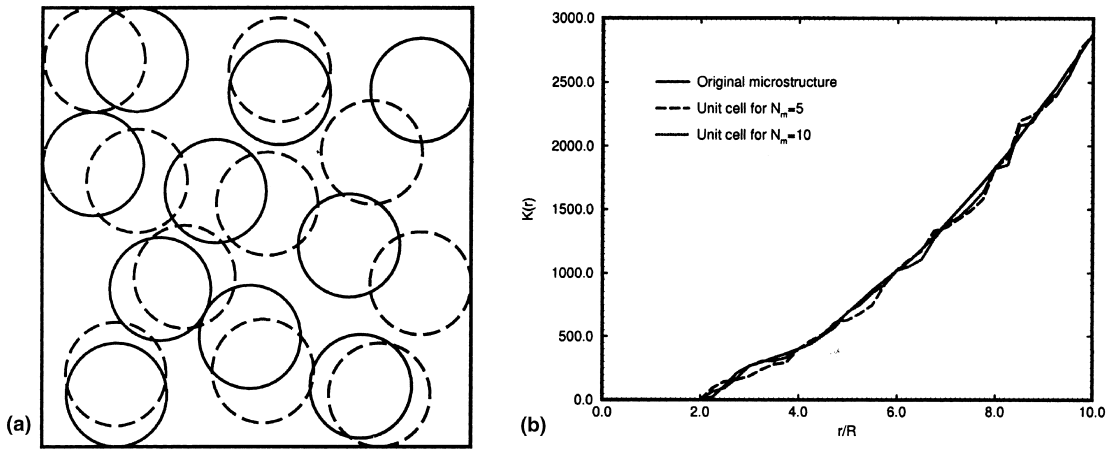

Fig. 16. Periodic unit cells with corresponding second-order intensity functions.

functions agree well at sampled points. Unfortunately, a significant deviation is distinct in all other points. To improve coincidence of both functions we increased a number of sampled points to 10 ($N_m = 10$) and used solutions obtained for $N_m = 5$ as our initial guesses. In this way, a much better agreement was attained, see Fig. 16. Fig. 16 shows how the unit cell evolved when increasing an accuracy of our solution. It is interesting to note that when running for example the AUSA for $N_m = 10$ from the beginning, we arrived at the minimum equal to 1.63×10^{-4} after approximately 173,000 function evaluations. However, when starting with $N_m = 5$ and then continuing with $N_m = 10$, we received the desired minimum after 115,000 function evaluations only. This just confirms similar conclusions drawn from experiments with complicated functions.

Acknowledgements

Financial support was provided by the GAČR 103/97/1255 and GAČR 103/97/P040 grants, by the research project J04/98:210000003, and by the IGČVUT 3099K1322.

References

[1] M. Axelsen, Quantitative description of the morphology and microdamage of composite materials, Ph.D. Thesis, Aalborg University, 1995.
[2] J.E. Baker, Reducing bias and inefficiency in the selection algorithm, in: J.J. Grefenstette (Ed.), Proceedings of the Second International Conference on Genetic Algorithms, 1987, pp. 13–21.
[3] D. Beasley, D.R. Bull, R.R. Martin, An overview of genetic algorithms: Part 1, fundamentals, University Comput. 15 (2) (1993) 58–69.
[4] D.E. Goldberg, Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[5] D.E. Goldberg, Sizing populations for serial and parallel genetic algorithms, in: J.D. Schaffer (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 70–79.
[6] C.R. Houck, J.A. Joines, M.G. Kay, A Genetic Algorithm for Function Optimization: A Matlab implementation, North Carolina State University, available at www.fmmcenter.ncsu.edu/fac_staff/joines/papers/gaot.ps, 1995.
[7] V. Kvasnička, Augmented-simulated annealing adaption of feed-forward neural networks, Neural Network World 3 (1994) 67–80.
[8] S.T. Mau, A refined laminated plate theory, J. Appl. Mech. (1973) 606–607.
[9] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer, Berlin, 1992.
[10] Z. Michalewicz, T.D. Logan, S. Swaminathan, Evolutionary operators for continuous convex parameter spaces, in: A.V. Sebald, L.J. Fogel (Eds.), Proceedings of the Third Annual Conference on Evolutionary Programming, 1994.
[11] G.L. Povirk, Incorporation of microstructural information into model of two-phase materials, Acta Metall. Mater. 43 (8) (1995) 3199–3206.
[12] J. Procházka, EUROCODE 2, ČSN P ENV 1992-1-1, PROCON, 1995.
[13] B.D. Ripley, Modelling spatial patterns, J. Roy. Stat. Soc. Series B 39 (1977) 172–212.
[14] M. Šejnoha, K. Matouš, Nonlinear analysis of initially prestressed laminates, Acta Polytechnica, Submitted.
[15] D. Yuret, From genetic algorithms to efficient optimization, Master Thesis, Massachusetts Institute of Technology, Artificial Inteligence Laboratory, 1994.