Chapter 6

Structural optimization

Grisham's algorithm is attractive in optimization, since the computational effort required per function evaluation is relatively low. The algorithm can easily be combined with a well known but simple optimization algorithm, namely the genetic algorithm (GA).

In this chapter, the example from Bruhn [8] considered in Chapter 2 is therefore optimized with respect to minimum mass, using a μ-GA with a binary representation (e.g. see Carroll [15]). While a genetic algorithm is not necessarily the best algorithm for the problem under consideration, it is selected here, as to easily allow for discrete design variables in future.

The development is as follows: Firstly, the optimal design problem is formulated. Next, the genetic algorithm is briefly outlined, whereafter the differences between a GA and a micro-genetic algorithm (μ-GA) are described. (A μ-GA is based on, and is derived from, the GA.) In the following, the presentations by Groenwold [16] and Bolton [17] are closely followed. The chapter concludes with optimal design results for the example problem.

6.1 Objective function and constraints

The optimal design problem we consider in this chapter is formulated as follows: Find the minimum f^* such that

$$f^* = f(\boldsymbol{x}^*) = \min f(\boldsymbol{x}) \tag{6.1}$$

subject to the general inequality constraints

$$g_j(\boldsymbol{x}^*) \leq 0, \ j = 1, 2, \ldots, m \tag{6.2}$$

where $\boldsymbol{x}$ is a column vector in $\mathbb{R}^n$ and f and g_j are scalar functions of the design variables $\boldsymbol{x}$. $\boldsymbol{x}$ is subject to the subsidiary conditions $x_i^l \leq x_i \leq x_i^u$, with x_i^l and x_i^u respectively representing prescribed lower and upper bounds on x_i.

(6.1) is denoted the objective function, and frequently represents mass, exposed area, cost, etc. Constraints (6.2) may represent a limit on displacement, stress, strain, magnetic flux, etc. In finding a low value for (6.1), (6.2) may never be violated. For example: when the mass of an aircraft structure is minimized, the stresses in the members may never rise above

a prescribed value. Else a very light structure is found, which however fails immediately upon use.

6.2 The genetic algorithm (GA)

Genetic algorithms [18] are stochastic implicit enumeration methods based on Darwinism and in particular, on the natural theory of survival of the fittest. In brief, genetic algorithms attempt to improve the fitness of designs (expressed in terms of a scalar objective function) in consecutive generations. The initial generation is populated in a random fashion with chromosomes representing possible discrete designs. The genetic operators of selection, cross-over and mutation are then used in a controlled random manner to ensure that fit parents have a high probability of passing fit genetic material to their off-spring.

Even though GA's are infamous for high computational costs (a large number of function evaluations), GA's are attractive to engineers, since their construction is quite simple. Numerous applications of GA's in engineering optimization have been presented. In addition, GA's are also suitable for implementation on massively parallel processing machines and lend themselves to be tailored according to the behavior of the objective function under consideration [16].

The most important aspects of a GA are briefly described as follows [17]:

6.2.1 Representation of design variables

Consider an initial design population, constituting of e design vectors (or 'strings' in GA jargon) $\boldsymbol{x}$, created by a random selection of the variables in the variable space for each design.

The values of the variables in the strings must be represented by a unique coding scheme. We opt for binary coding, which is powerful and frequently used.

For example: the binary string (01101) of length $l = 5$ represents the real number 22:

$$0 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 + 1 \cdot 2^4 = 22$$

A real value x within bounds (x_b, x_e) is represented by binary coding in the following way:

$$x = x_{bin} \cdot \frac{(x_e - x_b)}{(2^l)} + x_b \tag{6.3}$$

where

$$x_{bin} = \sum_{i=1}^{l} z_i \cdot 2^{(i-1)} \tag{6.4}$$

and z_i can be either 1 or 0 with l the binary string length.

If the objective function has several variables, then the design vector can be represented by a concatenation of the coding of each variable [19]. For example, the three dimensional design vector

$$\boldsymbol{x} = [22\ 8\ 11]$$

with corresponding binary code (01101); (00010); (11010) is represented as

$$\boldsymbol{X} = [011010001011010]$$

(The uppercase symbol indicates that a concatenated string is considered.)

6.2.2 Selection

The selection operation selects e strings from the current population to form the mating pool. The strings corresponding to fit objective function values have the greatest chance to be selected for mating and hence to contribute to future generations. While a large number of different selection processes are possible, only the well known expected value selection, tournament selection and ranking selection are briefly discussed in the following.

Expected value selection (roulette wheel selection)

In the expected value selection [20], also known as the roulette wheel selection, the minimization problem is converted to a maximization problem by multiplying the objective function with -1. Also, the function values must be positive and therefore a constant must be added to functions with negative values. The relative fitness p_i for each design is calculated as follows:

$$p_i = \frac{f_i}{\sum_{i=1}^{e} f_i} \qquad i = 1, 2, 3, ...e \tag{6.5}$$

where f_i denotes the function value of design i. The cumulative probability space d_j is defined as:

$$d_j = \sum_{i=1}^{j} p_i \qquad j = 1, 2, 3, ...e \tag{6.6}$$

String i is selected for the mating pool if a random number v between 0 and 1 is generated and satisfies the condition: $d_{i-1} < v \leq d_i$ with $d_0 = 0$.

Tournament selection

Tournament selection simulates the process where individuals compete for mating rights in the population [15]. In the GA, e tournaments are held between a sub group of strings chosen randomly from the existing population. The design from each tournament with the lowest function value is selected for the mating pool.

Ranking methods

After ranking the strings in ascending order according to the objective function values, the relative fitness p_i of member i is expressed as

$$p_i = \frac{t_i}{\sum_{i=1}^{e} t_i} \tag{6.7}$$

where

$$t_i = 2 \cdot \frac{(e+1-i)^c}{(e^2+e)} \tag{6.8}$$

c is taken as any value between 1 and 10 (typically 1) and e is the population size. The cumulative probability space d_j is constructed using the above defined relative fitness p_i and the strings are selected as in the expected value selection.

6.2.3 Crossover

After selecting e strings for the mating pool, new designs are explored by the crossover process. Crossover allows selected individuals to trade characteristics of their designs by exchanging parts of their strings. The mating pool strings are randomly grouped into pairs and a breaking point in the strings for each pair is chosen randomly. The values at the string positions after the breaking point are interchanged between the pair and the new designs are copied to the new generation. Crossover for each pair is applied with a given probability p_c, usually between 0.6 and 1. For example, when crossover is applied at the third crossover position of the following strings

$$\boldsymbol{X}_1 = [011|0100]$$

$$\boldsymbol{X}_2 = [010|1101]$$

the strings exchange the last four bits and become:

$$\boldsymbol{X}_1 = [011|1101]$$

$$\boldsymbol{X}_2 = [010|0100]$$

If crossover for a pair is not applied, then the unchanged parents are copied into the next generation. Sometimes, only one child is produced per parents. Frequently, more than one braking or cross-over point is used during crossover.

6.2.4 Mutation

The mutation operation protects against complete loss of genetic diversity by randomly changing bit values in a string. For each bit in the population a random number is generated and the bit value is changed if the random number is less than the prescribed probability of mutation p_m. For example, if mutation occurs at position four of the following string

$$\boldsymbol{X}_1 = [011\underline{0}100]$$

the string becomes

$$\boldsymbol{X}_1 = [011\underline{1}100]$$

For an integer alphabet, the mutated value can randomly be selected from the possible values from the alphabet (jump mutation), or given the value of a neighbor (creep mutation). In a binary representation, prescribing the position has a similar effect (see the example in Section 6.2.1).

Mutation typically occurs at low probability, else convergence can be impaired.

6.2.5 Other operators

Numerous other operators have previously been proposed. It is outside the scope of this study to name them all. However, one further operator frequently encountered is 'elitism'. It is described as follows: At any stage of the GA search, the best string found to date may (permanently) disappear from the mating pool, due to the random nature of selection. If one opts for an implementation that returns the best string found at some stage to the mating pool, (after say every a generations), this process is known as elitism. Frequently, a value of unity is used for a.

6.3 μ-GA

A μ-GA is an implementation of the GA in which rebirth replaces mutation. The basic idea is to use very small populations (say 5 individuals). It is then hoped that the small population converges very quickly, whereafter 1 or at most 2 individuals are retained using tournament selection. The remaining individuals are then killed off, and randomly repopulated, a process called rebirth.

Rebirth was first proposed by Galante [21]. The implementation of Carroll [15] is similar to that of Galante, except that Galante used larger generations than the norm in the μ-GA. In addition, Galante retained mutation, which probably is superfluous.

While the 'free lunch' theorem [22, 23] effectively prohibits a general, exhaustive comparison between the GA and the μ-GA, the latter has a secondary advantage: A reasonable solution may sometimes be obtained relatively quickly, due to the small generations typically employed. This may be desirable when evaluation of the objective function is computationally very expensive, (e.g. when using non-linear finite element analyses), and when a visual inspection of the 'current best' solution can be used to defend additional computational effort.

An inherent drawback of a μ-GA is that premature convergence is likely. Terminating a population is obvious: it can for example be done when a prescribed fraction (say 0.8) of the population has converged to a given value. However, knowing when enough restarts have been made, is not simple, and will probably be influenced by factors like computational effort and the quality of the solution found to date.

6.4 Optimization of verification example

The minimum-mass design of the stiffened shear webs is taken as the objective function, while the design variables represent the dimensions of the 6 panels. The single constraint g considered in the optimization phase is the maximum allowable prescribed post-buckled stress, expressed as

$$g = \tau_{xy}^{max} - \tau_{xy}^{pres} \tag{6.9}$$

where τ_{xy}^{max} represents the maximum shear stress in the webs. τ_{xy}^{pres} represents the prescribed, maximum allowable post-buckled stress. The objective function is then formulated as

$$f = \sum_{k=1}^{l} m_k + \mu\rho(g)^2 \quad (\rho > 0 \text{ and prescribed}) \tag{6.10}$$

where l represents the number of structural members, m the mass of member k and

$$\mu = \begin{cases} 0 & \text{if } g \leq 0 \\ 1 & \text{otherwise} \end{cases} \tag{6.11}$$

For the current example the constraint is simply taken as the yield stress of the material: $\sigma_{yt} = 483$ MPa for both the web material, the flange and upright material.

Further to the single considered constraint on stress, a large number of bound constraints are included to ensure validity of the solution. These constraints are all expressed in terms of the relationships given in Grisham's method, and are included in Tables 6.1 and 6.2 which tabulate the numerical results.

The example is optimized using two different sets of design variables. Firstly, four geometric variables are selected, whereafter eleven geometric variables are used. With reference to Figure 2.1, the design variables for Case 1 are:

- the upper flange area,
- the lower flange area,
- the upright area (all equal), and
- the web thickness (all equal).

For Case 2, the variables are:

- the upper flange area,
- the lower flange area,
- the area of uprights number 2 through 6 (all different), and
- the web thickness of panels number 2 through 5 (all different).

In both cases, panels one and six are not considered because of the effects of the boundary conditions.

6.4.1 μ-GA parameters

In the implementation of the μ-GA, we use a small population size of 5, and a uniform cross-over with an 80% probability. One child per pair of parents is produced, and parent selection is based on tournament selection. Elitism is included.

Eccentricity of the uprights is taken into account. The cross-sections in the example are used; T-sections for the flanges and angle sections for the uprights. During the optimization phase, the finite element mesh is never updated or changed in any way from the initial model.

6.4.2 Optimal results

For Case 1, using the μ-GA, the optimum mass is obtained as 5.515 kg after 37 generations. This gives an 11.01% saving on mass from the original mass of 6.122 kg. This is highly

Variable	Description	Initial value	Final value	Bounds
x_1	Lower flange area [mm^2]	243.87	201.0	$100 \leq x_1 \leq 250$
x_2	Upper flange area [mm^2]	435.48	400.4	$200 \leq x_2 \leq 450$
x_3	Upright area [mm^2]	151.21	150.4	$50 \leq x_3 \leq 170$
x_4	Web thickness [mm]	0.635	0.5188	$0.3 \leq x_4 \leq 0.9$

Table 6.1: Case 1: Optimum results using the μ-GA

Variable	Description	Initial value	Final value	Bounds
x_1	Lower flange area [mm^2]	243.9	197.7	$100 \leq x_1 \leq 250$
x_2	Upper flange area [mm^2]	435.5	396.5	$200 \leq x_2 \leq 450$
x_3	Area of upright no. 2 [mm^2]	151.2	140.1	$50 \leq x_3 \leq 170$
x_4	Area of upright no. 3 [mm^2]	151.2	142.8	$50 \leq x_4 \leq 170$
x_5	Area of upright no. 4 [mm^2]	151.2	138.5	$50 \leq x_5 \leq 170$
x_6	Area of upright no. 5 [mm^2]	151.2	141.1	$50 \leq x_6 \leq 170$
x_7	Area of upright no. 6 [mm^2]	151.2	141.9	$50 \leq x_7 \leq 170$
x_8	Panel 2: Web thickness [mm]	0.635	0.506	$0.3 \leq x_8 \leq 0.9$
x_9	Panel 3: Web thickness [mm]	0.635	0.494	$0.3 \leq x_9 \leq 0.9$
x_{10}	Panel 4: Web thickness [mm]	0.635	0.490	$0.3 \leq x_{10} \leq 0.9$
x_{11}	Panel 5: Web thickness [mm]	0.635	0.497	$0.3 \leq x_{11} \leq 0.9$

Table 6.2: Case 2: Optimum results using the μ-GA

significant in aircraft structures. At the optimum, the stresses are $\sigma_{mises} = 325.8$ MPa in the web and $\sigma_{mises} = 392.4$ MPa in the flanges.

The stress results of the optimized design for Case 1 are plotted in Figures 6.1, 6.2 and 6.3. Figures 6.4 and 6.5 show vector plots of the maximum and minimum principal stresses in the web.

For Case 2, the optimum mass is obtained as 5.366 kg after 23 generations. This gives a 14.08% saving on mass from the original mass of 6.122 kg. This is very similar to the four variable case. At the optimum, the stresses are $\sigma_{mises} = 307.7$ MPa in the web and $\sigma_{mises} = 427.8$ MPa in the flanges.

It is noted that neither Case 1 nor Case 2 are converged, since neither the stress constraint nor the variable bounds are active in either case. Case 2 does produce a more optimal result.

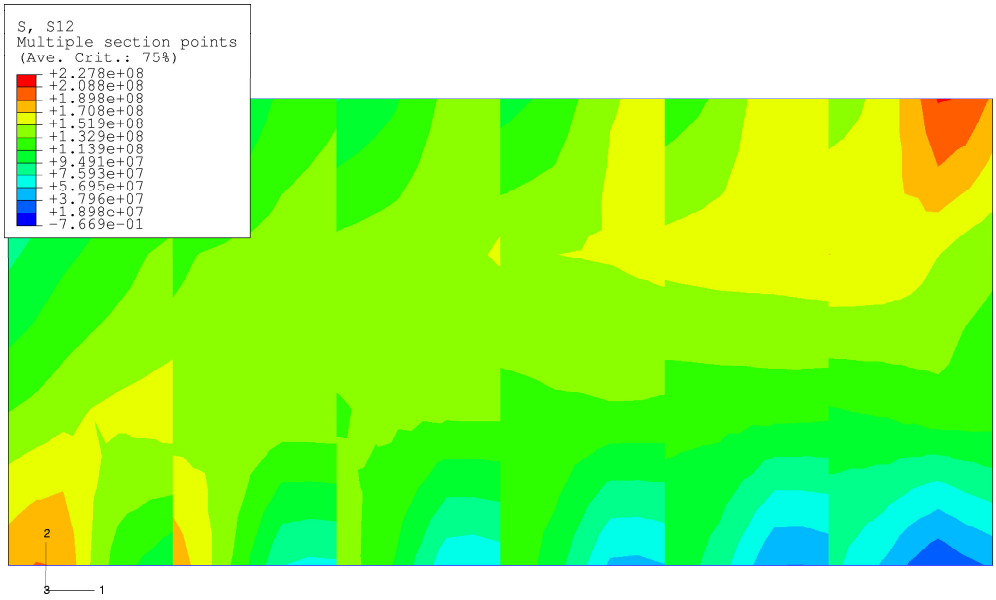

Figure 6.1: Unsmoothed shear stress distribution (τ_{xy}) in the web after the final iteration

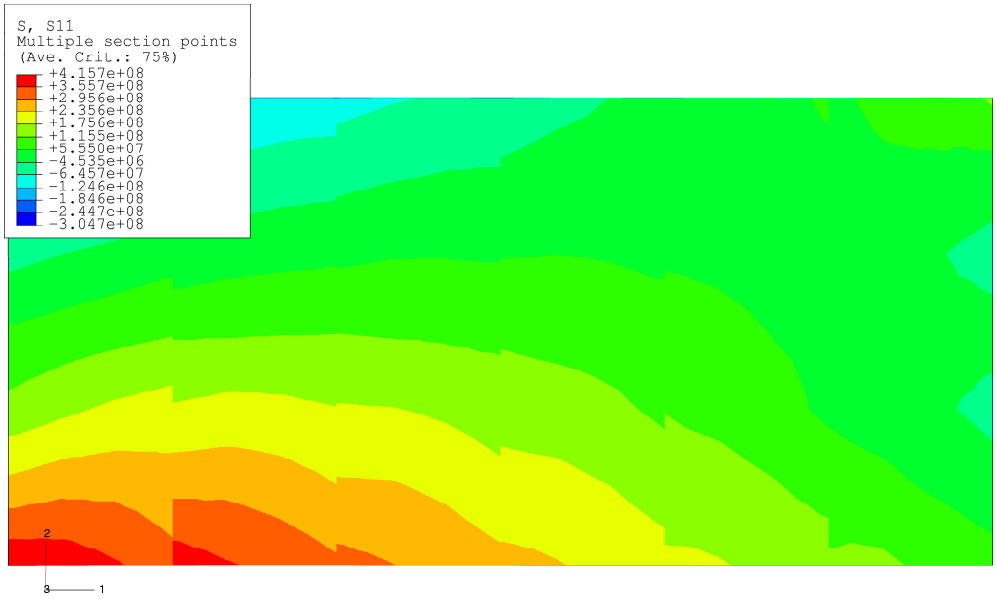

Figure 6.2: Unsmoothed normal stress (σ_x) in the web after the final iteration

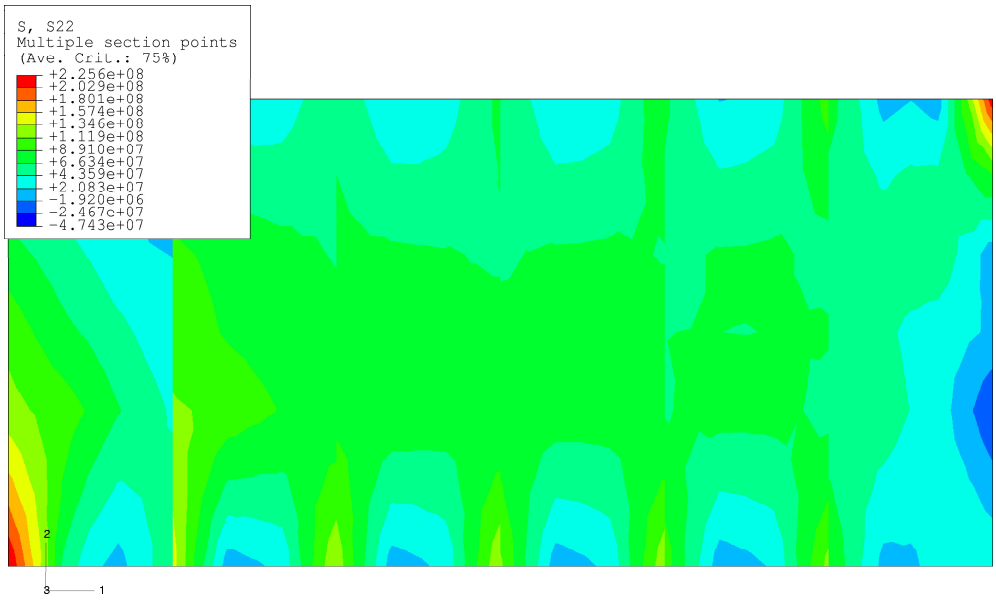

Figure 6.3: Unsmoothed normal stress (σ_y) in the web after the final iteration

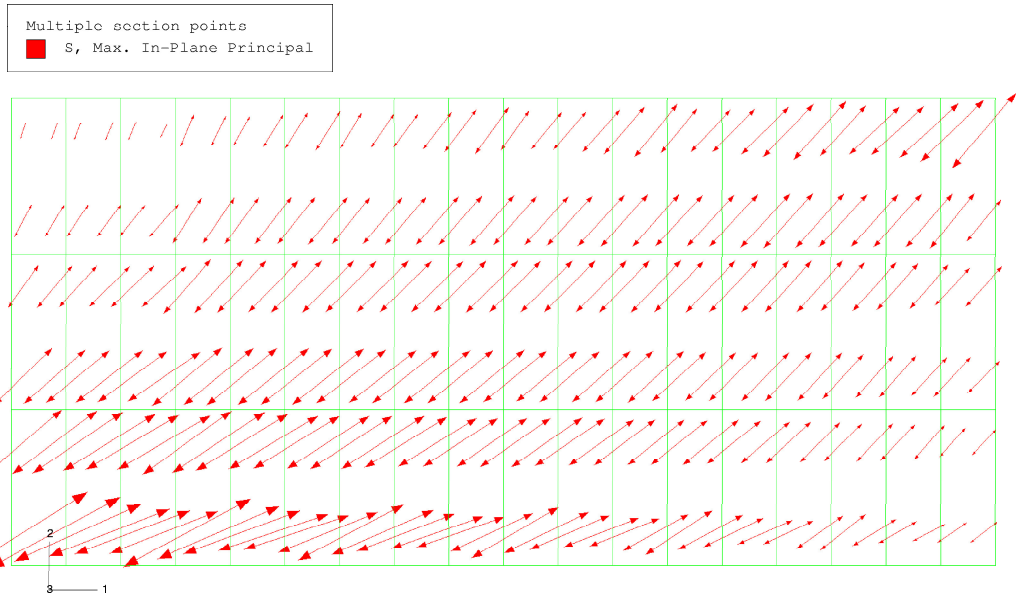

Figure 6.4: Maximum principal web stress vector plot after the final iteration

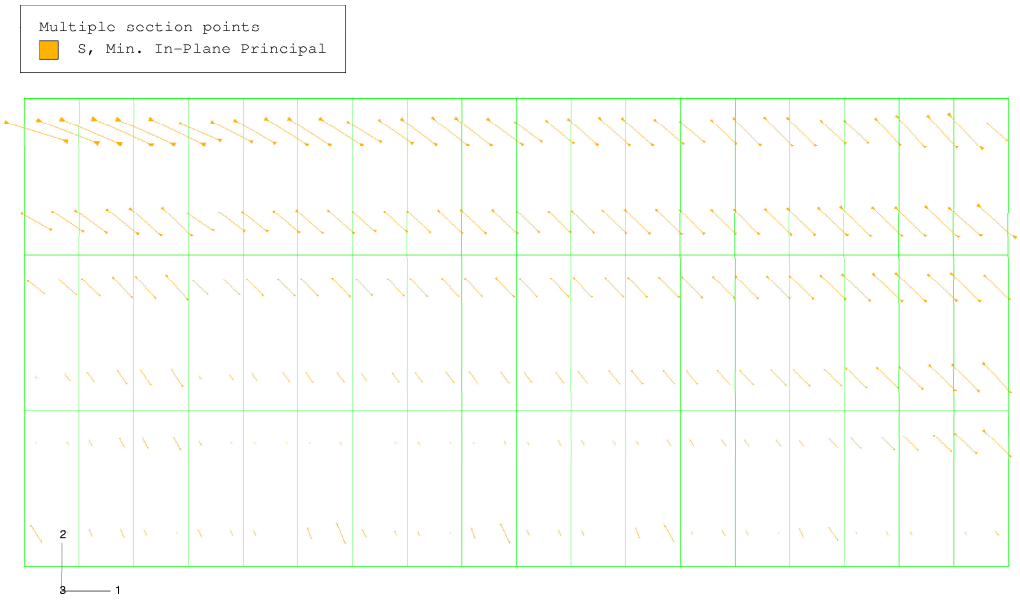

Figure 6.5: Minimum principal web stress vector plot after the final iteration