Applying genetic algorithms to optimization of reinforced concrete beam

Matěj Lepš
advisor Zdeněk Bittnar
CTU, Fac. of Civil Eng., Dep. of Structural Mechanics
Thákurova 7, 166 29 Prague 6

Abstract

This paper outlines application of genetic algorithms (GAs) to a class of optimization problems associated with various 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.

Keywords: optimization, genetic algorithm, reinforced concrete beam

1 Introduction

A wide spread 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 non-linearities 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.

The paper is organised as follows. Formulation of the design problem is investigated in Section 2. Some techniques used for optimization are described in Section 3. Section 4 presents description of two genetic algorithms. The next Section contains example results showing the power of genetic algorithms in effective search for the desired solution.

2 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

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

$$M_{Sd} \leq M_{Rd} \,. \tag{3}$$

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. Inqs. (2) and (3) express selected design criteria according to EURODODE 2 standard (EC2) [7] for reinforced concrete (RC) structures. In particular, Inq. (2) represents the serviceability requirement, where l is the span and δ_{lim} is the maximum permissible deflection of a beam. Condition for the cross-section bearing capacity, Inq. (3), given in terms of moments deserves more attention.

Consider a representative cross-section of reinforcement 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 [3].

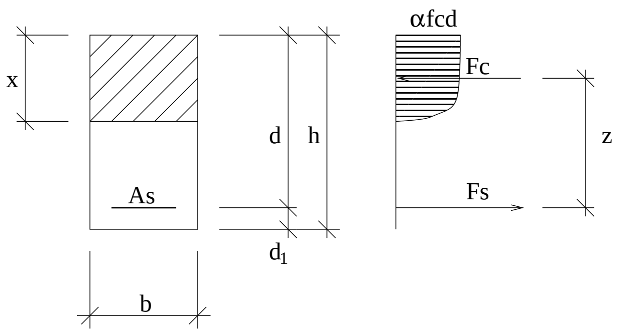


Figure 1: RC beam with lower reinforcement

According to [7] 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

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

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 constraint violations

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

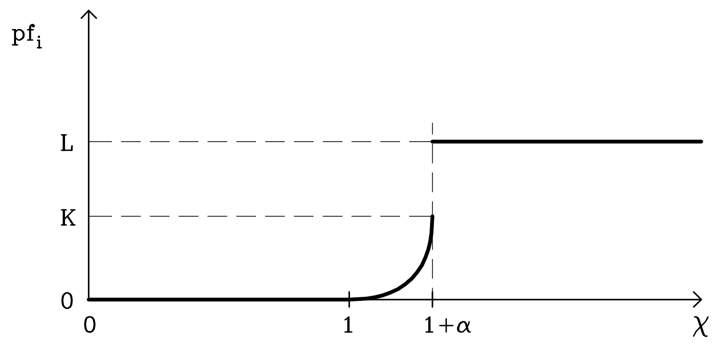


Figure 2: Penalty function

In our present approach the penalty functions pf_i assume in general the form displayed in Fig. 2. Consider a parameter χ

$$F_i \leq F_{i,max} \ , \qquad\qquad \chi = \frac{F_i}{F_{i,max}} \quad . \tag{7}$$

Then in a closed interval $< 0, 1 >$ the form pf_i is equal to zero, in interval $< 1 + \alpha, \infty >$ pf_i is assigned to the user defined parameter L and in interval $< 1, 1 + \alpha >$ this function is provided by

$$pf_i = K\Big(\frac{\chi - 1}{\alpha}\Big)^{\beta} \ , \tag{8}$$

where α, β and K are the user defined parameters of the proposed penalty function. Usually a large number is assigned to parameter β whereas α approaches near zero. Parameter K is lower than or equal to L.

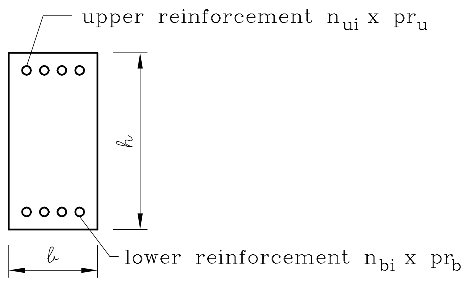
At this point, however, we should warn the reader against perceiving the above approach as a general one for solving a constrained media problem. Although for a moderate number of constraints the approach, which includes penalties in the function evaluation, may prove to be reliable and efficient, the 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 strandards) it appears reasonable to follow for example an approach outlined in [6].

3 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 a 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.

3.1 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. 3,



upper reinforcement n_{ui} x pr_u
h
b
lower reinforcement n_{bi} x pr_b

Figure 3: Beam cross-section

where h and b represent the height and width of the cross-section, respectively. Table 1 lists 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 0.025m apart or it can change continuously.

Table 1: Dimensions of the cross-section

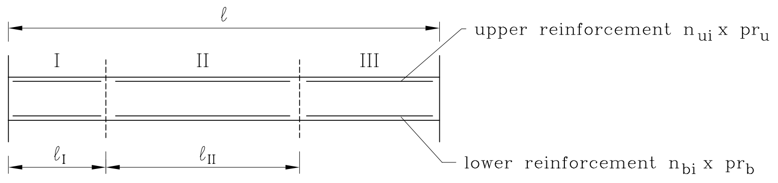
Variable	Units	Minimum	Maximum	Precision	Comment
h	[m]	0.15	0.85	0.025	Discrete values
		0.15	0.85	0.001	Continuous values
b	[m]	0.15	0.45	0.025	Discrete values
		0.15	0.45	0.001	Continuous values

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 ten parameters selected to control an amount and location of the bending steel reinforcement. Note that all variables are treated as integer numbers.

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

In Table 2 pr_u and pr_b represent upper and lower reinforcement along the whole span of a beam, see Fig. 4. 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. 4. For the design purposes, 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 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.



ℓ
I
II
III
upper reinforcement n_{ui} x pr_u
lower reinforcement n_{bi} x pr_b
ℓ_I
ℓ_{II}

Figure 4: Beam sections

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

$$min_i \leq x_i \leq max_i \;, \tag{9}$$

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 is desirable, then $p_i = 0.00001$. If $D_i \subseteq N_0$ (initeger 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. 11. Each variable x_i can be transformed into a nonnegative integer number $y_i \in N_0$ as

$$y_i = \left[\frac{x_i - min_i}{p_i}\right] . \tag{10}$$

An inverse transformation is given by

$$x_i = y_i \, p_i + min_i \; . \tag{11}$$

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

$$\frac{max_i - min_i}{p_i} \leq 2^k \; . \tag{12}$$

An integer number k is provided by

$$k = \left[\frac{\ln\left(\dfrac{max_i - min_i}{p_i}\right)}{\ln 2}\right], \tag{13}$$

where operator $[z]$ denotes the integer part of z. It can be easily recognized that 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 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.

4 Genetic algorithms

Genetic algorithms 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 a 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.

```
1  $t = 0$
2  generate $P_0$, evaluate $P_0$
3  **while** (**not** termination-condition) {
4     $t = t + 1$
5     select $M_t$ from $P_{t-1}$                  (apply sampling mechanism)
6     alter $M_t$                                  (apply genetic operators)
7     create $P_t$ from $M_t$ and evaluate $P_t$   (insert new individuals into $P_t$)
8  }
```

Algorithm 1: Principle of genetic algorithm

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 controling the step 6. Steps 5 and 7 will be explained in more details when formulating various algorithms we tested.

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 idividuals are selected for mating this operator works in accord with Fig. 5.

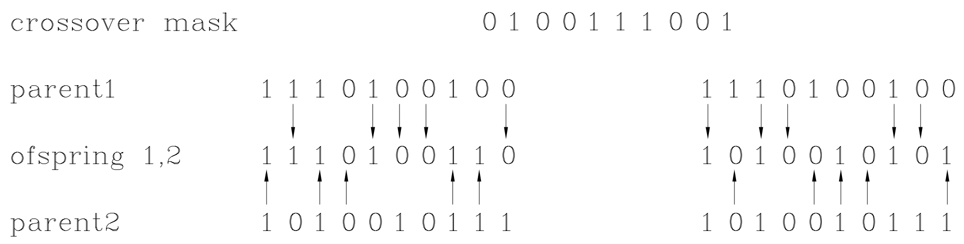


Figure 5: Uniform crossover

First, a random mask of the same length as the parents is generated. To create *offspring*−1 we proceede 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. 6, 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 point after which parents exchange their tails. Crossover operator is usualy applied with only a certain probability p_c. Therefore, not all pairs of individuals selected for mating are modified by the crossover step.

0 0 0 0 0 0 1 1 1 1   0 0 0 0 0 1 1 1 0 0

a)   b)

Figure 6: a) Single-point crossover mask, b) Two-point crossover mask

However, they still are bound to be disrupted by *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 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 exist several other perhaps more appealing approaches, which deal with this so called "premature convergence" towards a local minimum [5].

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 probabillity, Fig. 7.

$1\ \overset{\downarrow}{1}\ 1\ 0\ 1\ 0\ \overset{\downarrow}{0}\ 1\ 1\ 0 \qquad\qquad 1\ \overset{\downarrow}{0}\ 1\ 0\ 1\ 0\ \overset{\downarrow}{1}\ 1\ 1\ 0$

Figure 7: Mutation

It is generally accepted that mutation plays a secondary role in a process of recombination, and as in nature the likelihood of its appearence 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.

Genetic algorithm I (GAB I)

To keep our promise given in the introductory part 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 details:

Step 5 Individuals selected for reproduction are copied to the "mating" pool according to their relative performance reffered 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

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

where N is the number of individuals in a population and f_i is the function value associated with the i^{th} 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**) [1], [4], [6]. This method allocates individuals according to their integer part of e_i. The remaing places in a population are then sampled according to their fractional part using a spinning roullete wheel. The fractional parts represent a succes probability of selection. After each spin, the expected value of the selected individuals is set equal to zero.

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 GA cycles all individuals start to look alike and the algorithm 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 [2].

Steps 6&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 individulas in P_{t-1} by a new offspring to create a new generation P_t.

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 then those selected for dying. Otherwise, there are no changes introduced in population P_{t-1}.

5 Examples

As an example we selected a continuous beam subjected to a uniformly distributed load according to Fig. 8. Due to symmetry, only one half of the beam was analysed. Distribution of internal forces (bending moment and shear force) were found using the finite elemet method [3]. 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.

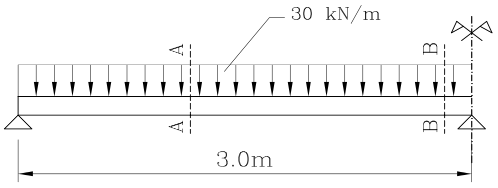


Figure 8: Continuous beam subjected to uniform loading

To test 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 through out the GA run. Optimization process was terminated when there was no change in the best individual fitness observed over a certain number GA cycles. Results appear in Tables 3-5.

Table 3: Example 1

Values	b [mm]	h [mm]	Price [Kč]
Continuous	150	294	898.48
Discrete	150	300	906.39

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. 9 (hollow circle). Function $f(b, h)$ in Fig. 9 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).

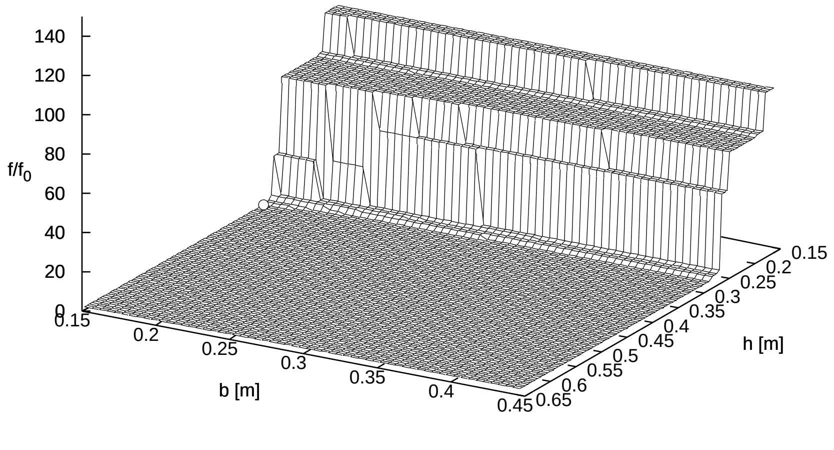


Figure 9: Distribution of the objective function

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 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 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 lgorithm **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 a 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 next generations, which essentily reduces an oportunity 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. 10 displays convergence characteristics of both algorithms.

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

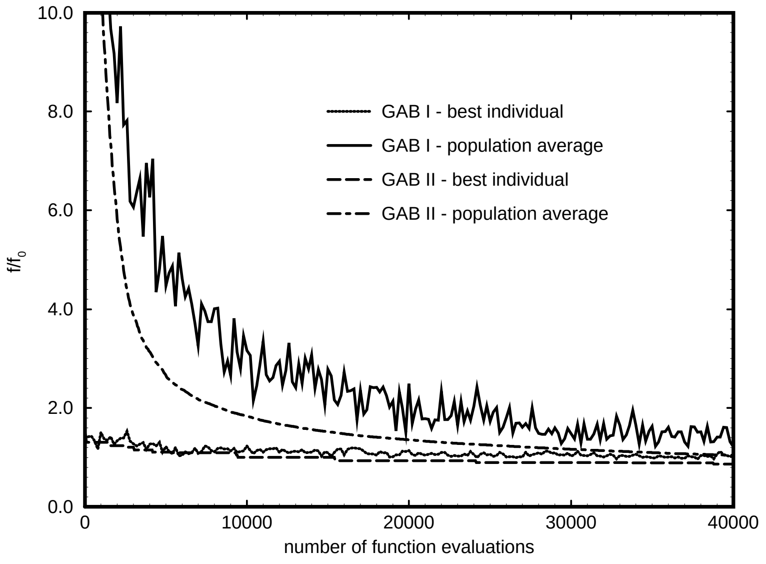


Figure 10: Distribution of the objective function

Finally, Table 5 summarises results obtained from five independent runs using the steady state genetic algorithm. It shows variation in both dimensions and profiles of the lower (section A-A) and upper (section B-B) reinforcement bars, which yet manifests the discrete nature of the problem.

Table 5: Example 2 - results from 5 independent runs using **GAB II**

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

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 standars. Complying with additional design requirements just increases a number of constrains, which eventualy prevents the presented algorithms from working. Our early experiments, however, suggest the so called *Augmented simulated annealing* is the right method of attack [5]. Another possibility is to let genetic algorithms to do the hard work when exploring promissing areas in the solution space initially and then call the local optimizer to descent individual hills in search for the best solution. Those are two routs we are currently pursuing.

Acknowledgments

Financial support was provided by the GAČR 103/96/1271 and GAČR 103/97/P040 grants.

References

[1] Baker, J. E. : *Reducing bias and inefficiency in the selection algorithm*, Proceedings of the Second International Conference on Genetic Algorithms, Grefenstette J.J. editor, pp. 13-21, **1987**.

[2] Beasley D., Bull D.R., and Martin R.R.: *An overview of genetic algorithms: Part 1, fundamentals*, University Computing, 15(2), pp. 58-69, **1993**.

[3] Bittnar Z., Šejnoha J. : *Numerické metody mechaniky*, ČVUT, Praha, **1992**.

[4] Goldberg D. E. : *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley, **1989**.

[5] Kvasnička, V. : *Augmented simulated annealing adaption of feed-forward neural networks*, Neural Network World, Vol. 3, pp. 67-80, **1994**.

[6] Michalewicz Z. : *Genetic Algorithms + Data Structures = Evolution programs*, Springer-Verlag, **1992**.

[7] Procházka J. : *EUROCODE 2*, ČSN P ENV 1992-1-1, PROCON, **1995**.