Abstract

When it comes to solving nonconvex, discontinuous, or discrete problems in Structural Optimization (e.g. maximizing first eigenfrequency of a structure), the use of computationally expensive Genetic Algorithms (GA’s) gets interesting. GA’s are stochastic optimization algorithms based on natural selection and genetics. In contrast to traditional gradient-based methods, GA’s work on populations of solutions which evolve typically over hundreds of generations. A tool is presented, which applies GA’s to solve typical problems in structural optimization, integrating ANSYS on a UPF (User Programmable Features) level to evaluate the objective function (fitness values of GA individuals). To overcome efficiency limits, the method is implemented for parallel evaluations on a workstation cluster. The performance of the software tool is shown by two real world applications, the frequency optimization of a complex machine-tool frame and the weight minimization of a fuel cell plate.

1 Introduction

In the past, the intuition and experience of engineers played the key role in designing structures. Recent years have seen the development of numerical tools, which provide conceptual designs for a given design space and specified boundary conditions. The aim of these tools is to support the intuition and the experience of an engineer.

Most of these tools are based on gradient methods, which work well and very efficient for continuous and convex objective functions as e.g. compliance minimization. An example for a highly efficient gradient based method is the topology optimization method using homogenization introduced by Bendsoe and Kikuchi [2], as it is implemented in ANSYS.

The motivation to develop the presented Genetic Algorithm (GA) based tool in the optimization group at the Institute of Mechanical Systems (IMES@ETHZ), comes from the ongoing PhD project of Marion Uebersax [8]. She develops methods to optimize the dynamic behaviour of machine tools. There it is interesting to compare the efficient but sensible gradient based methods with a more robust Genetic Algorithm based approach. Another motivation lays in the diploma thesis of Oliver König [6] where a Genetic Algorithm tool was developed to optimize multi-material structures. This GA procedure used an own FE code to evaluate the objective functions.

These two projects led to the diploma thesis of Roman Gätzi [3] with the goal to extend the existing GA procedure to a general purpose Structural Optimization tool as it is presented in this paper. To be able to easily adapt the tool for all kind of Structural Optimization problems, ANSYS is integrated to evaluate the objective functions.

The paper is organized as follows. In Section 2 the basic concepts of Genetic Algorithm’s are outlined by a simple example and it is shown how they can be used in Structural Optimization. Section 3 describes the optimization tool developed, focused on the integration of ANSYS into the GA procedure. The performance of the
algorithms is shown in Section 4 and 5, where two real world structures are optimized for completely different objectives.

2 Genetic Algorithms

Genetic Algorithms can be described as search algorithms based on the mechanics of natural selection and natural genetics. They belong to a category of stochastic search methods, with an additional strength that randomized search is conducted in those regions of the design space which offer the most significant potential for gain. The primary references on the topic are Holland’s “Adaption in Natural and Artificial Systems” (referenced in [4]) and the book from Goldberg [4]. With focus to structural optimization, the paper [5] by Hajela gives a good introduction.

GA’s are not severely limited by discontinuous design spaces like techniques derived from mathematical programming principles. On the other side there is usually a stiff computational requirement associated with the use of GA’s. Therefore genetic algorithms represent a good solution approach for design problems where standard mathematical programming techniques are inefficient. The main advantages of GA’s can be formulated as follows:

- GA’s do not require function derivatives.
- GA’s proceed from several points in the design space, this makes it more likely to find global optima.
- GA’s work on a coding of the design variables. This allows them to work in design spaces consisting of a mix of continuous, discrete, and integer variables.

2.1 The Concept of Genetic Algorithms

The concept of GA’s is introduced in this section. The explanations are closely related to the book of D. Goldberg and therefore referenced only once [4]. The Genetic Algorithm is step by step applied to the following example:

Maximize \( f(x) = x^2, \quad x \in [0, 31] \) \hspace{1cm} (1)

Coding. The first step in solving an optimization problem with genetic algorithms is the coding of the design variables. The design variables need to be coded as a finite-length string over a finite alphabet. For the example above, \( x \) is coded as a binary unsigned integer of length 5. It can thus take on numbers between 0 (00000) and 31 (11111).

Initialization. Since genetic algorithms do not start only from a single point in the design space, but from a population, the initial set has to be created at random. Table 1 reflects the fitness function (objective function) evaluated for each chromosome (each string), with a population size 4 (strings 1 to 4).

<table>
<thead>
<tr>
<th>Str. No.</th>
<th>Initial Pop.</th>
<th>( x )</th>
<th>( f(x) )</th>
<th>( p_{select} )</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01101</td>
<td>13</td>
<td>169</td>
<td>0.14</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>11000</td>
<td>24</td>
<td>576</td>
<td>0.49</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>01000</td>
<td>8</td>
<td>64</td>
<td>0.05</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10011</td>
<td>19</td>
<td>361</td>
<td>0.31</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>1170</td>
<td>1.00</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Sample Problem: Initial Population and Reproduction

Reproduction. Reproduction is the first GA operator to apply to the population. It is a process in which individual strings are copied according to their fitness function values. This means strings with a higher value have a higher probability of contributing one or more offspring in the next generation. A simple way to implement this operator is the weighted roulette wheel method. The offsprings are allocated a value using a roulette wheel with slots sized according to the percentage of fitness. For example string 1 is given 14.4 % of the roulette wheel. All the offsprings can be generated with a simple spin of the wheel. Once a string has been selected for reproduction, an exact replica of the string is made. This string is then entered into a mating pool, a tentative new population, to be used by the next genetic operator. See tables 1 and 2 for the sample problem.

Crossover. After reproduction, simple crossover proceeds in two steps. First, members
of the newly reproduced strings in the mating pool are mated at random. Second, each pair of strings undergoes crossover as follows: an integer position $k$ along the string is selected uniformly at random between 1 and $l - 1$, where $l$ is the string length. Two new strings are created by swapping all characters between position $k + 1$ and $l$ inclusively. For example consider string 1 and 2 from the mating pool in Table 2 for $k = 4$:

$$
\begin{align*}
A_1 &= 0110|1 \implies \tilde{A}_1 = 01100 \\
A_2 &= 1100|0 \implies \tilde{A}_2 = 11001
\end{align*}
$$

Table 2 shows the new population and its fitness values after crossover.

<table>
<thead>
<tr>
<th>Str. No.</th>
<th>Pool after Repr.</th>
<th>Mate/ X-over Site</th>
<th>New Pop.</th>
<th>Fitness $f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0110</td>
<td>1</td>
<td>2/4</td>
<td>01100</td>
</tr>
<tr>
<td>2</td>
<td>1100</td>
<td>0</td>
<td>1/4</td>
<td>11001</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>000</td>
<td>4/2</td>
<td>11011</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>011</td>
<td>3/2</td>
<td>10000</td>
</tr>
</tbody>
</table>

Table 2: Sample Problem Crossover and New Fitness Values

**Mutation.** Mutation is the last operator for a simple genetic algorithm. Mutation is the occasional random alteration of the value of a string position. For binary strings this simply means changing a 1 to 0 or inverse. For the sample problem the probability of mutation is assumed to be 0.001. With 20 transferred bit positions there should be $20 \cdot 0.001 = 0.02$ bits to undergo mutation during a given generation. This indicates that no bits undergo mutation for this probability value.

The mutation operator finishes one cycle of a genetic algorithm. For the sample problem the new generation can be found in Table 2. The process starts again by evaluating the fitness values for each chromosome. Notice how both average and maximal performance improved in the new population.

In the following some additional GA features which are used in the project are briefly introduced:

- **Elitist Strategy.** This strategy ensures that the best individuals stay in the population.
- **Overlapping Populations.** The pool of individuals before reproduction consists of a GA with overlapping populations in the previous population and a specific amount of new individuals. The worst individuals of the entire pool are removed in order to return the population to its original size. Since only part of the population is generated, this strategy saves computation time.

### 2.2 Parameters in Genetic Search

The initiation of genetic search requires specification of some key parameters:

- **Population Size:** $\text{popsize}$. The number of strings processed in each generation. Typical size for structural optimization problems: $25 - 125$.
- **Number of Generations:** $\text{ngen}$. Usually a value in the hundreds is needed to make sure that the solution has time to converge.
- **Crossover Probability:** $\text{pcross}$. Values ranging from 0.6 to 0.9 have been used in numerical experiments with very satisfactory results.
- **Mutation Probability:** $\text{pmut}$. Probability values between 0.005 and 0.05 produce in general good results.
- **Overlapping Gap:** $\text{prepl}$. The overlap parameter specifies how many percent of the population is created new for each generation. A typical value is 0.5.

### 2.3 Using Genetic Algorithms in Structural Optimization

To apply Genetic Algorithms to Structural Optimization problems two main aspects have to be considered.

**Discretization and Coding of the Design Variables.** Any kind of chromosome used as a representation of the design variables in GA’s is discrete. Therefore, the design variables have to be discretized and a mapping from the design domain to a chromosome has to be defined.

Example: If plate thicknesses of a sheet metal structure shall be optimized, the design variables would be a discrete set of thicknesses (probably represented by real constant sets in a FE model). An individual of the GA would then consist of...
a string of integer numbers, where each integer value denotes a certain sheet thickness.

**Evaluating the Fitness Function.** A GA needs a function which determines the fitness of each individual in a population. In a structural optimization this function often includes a Finite Element Analysis. For a typical GA evaluation, the fitness has to be computed for thousands of individuals. Therefore it is imperative that the FE evaluations have to be fast and well integrated in the GA environment.

For the example of the sheet metal structure, a typical objective would be to maximize stiffness of the structure subject to a given weight.

### 3 Optimization Tool

The structural optimization tool developed in this project is based on the parallel Genetic Algorithm library PGAPack by David Levine (Argonne National Laboratory). It provides all the needed GA functionalities and allows to run the optimization in a parallel master-slave process on a workstation-cluster. The FEA package ANSYS is used to evaluate fitness functions of the GA as well as for pre- and postprocessing of the FE models.

Figure 1 shows the detailed optimization procedure, without parallelization.

As the flow chart already indicates, the main challenge in making this optimization tool work was to set up all the connections between GA, ANSYS and parallel evaluations. In the following important aspects of this work are described.

#### 3.1 Parallel Evaluation of the Fitness Function

A typical GA run needs thousands of fitness evaluations, which include in this project always a FE evaluation of the structure to be optimized. This fact makes it very interesting to use the parallel capabilities of the PGA library. Using the MPI library (Message Passing Interface) it allows to distribute the fitness evaluations and therefore the FE-evaluations on a workstation cluster. The parallelization is implemented in a master-slave hierarchy. The master process controls the GA and distributes the individuals of each generation to different slave machines to evaluate their fitness values.

#### 3.2 Fitness Evaluations Using ANSYS

ANSYS was chosen to evaluate the fitness functions of the individuals (structures) because of the following reasons.

- **ANSYS as Subroutine.** ANSYS allows it to be included as a subroutine in a Fortran program like the PGA main program.
- **Flexibility.** The use of ANSYS to evaluate fitness functions allows to optimize for every objective ANSYS can evaluate. This makes the optimization tool very powerful.
- **Pre- and Postprocessing.** ANSYS provides comfortable Pre- and Postprocessors which allow to easily create the discrete optimization models and display the optimization results.

**ANSYS as Subroutine.** In the following the main concepts of using ANSYS in a Fortran program are described. ANSYS can be called in a Fortran program as shown by the following code fragments. In addition it is shown how to read in the optimization model stored in a traditional ANSYS input file.

```fortran
c Include ANSYS Defs
#include "ansysdef.inc"
#include "impcom.inc"
...
c Initialization of ANSYS
   cmd = 'START_OF_USER'
   where = mainan(cmd)
...
c Read input file of the
   c optimization model and pass it to ANSYS
10 open(4, file='optmodel.inp')
   read(4, '(a)', end=999) cmd
   where = mainan(cmd)
   go to 10
999 continue
   close(4)
```

**Communicating between ANSYS and Fortran programs.** Once ANSYS is initialized there are mainly two ways to exchange data and commands between ANSYS and the Fortran-GA program. One can issue standard APDL commands (Ansys Parametric Design Language) as shown in the next code fragment where the database is saved.

```fortran
   cmd = 'SAVE,BEST_INDIVIDS,db'
   where = mainan(cmd)
```

The other possibility is to directly issue UPF commands (User Programmable Features) in Fortran. This native ANSYS interface allows faster access to the ANSYS database. The following line shows how pressure can be applied on element faces.

```fortran
call eprput(elem,face,8,pres)
```

**Compilation.** To finally compile the main optimization program, the Makefile ANSCUSTOM has to be modified in order to link all necessary libraries from PGA and MPI. With the created executable the master process of the GA optimization can be started. The master then starts all the slave processes on the specified workstation cluster.

4 Eigenfrequency Optimization of a Machine Tool

One of our current projects, at the center of technologies of the ETH Zurich, is the development of methods to optimize the dynamic behavior of machine tools [8]. The Figure 2 shows a machine tool of the company MIKRON Agno which must be optimized in order to maximize its fundamental frequency. The response of a structure to dynamic loading depends on the eigenfrequencies of the structure. Excessive vibrations occur when the frequency of the dynamic loading is too close to one of the eigenfrequencies of the structure. The problem is significant for machine tools because the vibrations influence the accuracy and the surface quality of the product. They also decrease the lifetime of the machine and of the cutting tools. They also affect the productivity and the operator comfort. In this section we describe the use of genetic algorithms to maximize the fundamental frequency of a complex structure.

4.1 Problem Description

**Objective and Constraints.** The objective function is to maximize the fundamental frequency of the structure by varying the thickness
of the shell elements. The optimization is subject to a weight constraint: The new structure should not be heavier than the original one.

$$\begin{align*}
\text{Maximize:} & \quad f_1(t_{el}) \\
\text{Subject to:} & \quad w \leq w_o
\end{align*}$$  
(3)  
(4)

**Discrete Design Space.** The model shown on the Figure 3 is used as start design for the genetic algorithm. This topology design was the result of a previous optimization procedure [9].

![Figure 3: Discretization of the structure](image)

The rotationally symmetric structure is discretized with approximately 5200 elements. The frame of the machine is modeled with 5032 2D-Shell elements. Additionally, 32 mass elements of 250 kg fixed on the structure with 128 1D-rigid elements are used to represent the processing units. We used a rather coarse mesh in order to reduce the time required per evaluation. For the boundary conditions, the machine is fixed at the bottom where neither translation nor rotation of the knots is allowed. The steel machine is 2500 mm high and has a diameter of 3125 mm.

### 4.2 GA Coding and Fitness Function

**Coding of the Design Variables.** Due to the symmetry of the machine, only one 16th of the structure is used as design space. This half sector is divided into 78 zones defining the design variables. Instead of modifying each element, the GA only changes the thickness of these zones. The genetic algorithm can choose the optimal thickness between 50%, 75%, 100%, 125% or 150% of original one. Each design variable can then take five different values, representing these ratio. An integer string with 78 alleles is used to code the set of design variables of the GA. The number of possible solutions can then be determined with:

$$N_{solutions} = 5^{78} \approx 3.3 \cdot 10^{54}$$  
(5)

**Fitness Function.** The objective function defined above has to be translated into a fitness function including the objective and the weight constraints. For the formulation of the unconstrained fitness function (6), the fundamental frequency is normalized in order to scale the maximum fitness value to 1000. In order to respect the weight constraint, the penalty function (7) is defined which is only active when the constraint is violated. This results in the final fitness function given by the expression (8).

$$\begin{align*}
\text{Score} & = \frac{f_1}{\text{score}_{\text{norm}}} \\
\text{Penalty} & = \begin{cases} 
(w - w_o) & \text{for } w \geq w_o \\
0 & \text{for } w < w_o
\end{cases} \\
F & = (1 - \text{penalty}) \cdot \text{score}
\end{align*}$$  
(6)  
(7)  
(8)

### 4.3 Results

The fundamental frequency of the optimized machine is 73% higher that of the initial structure. Additionally, the total weight decreases to 4.5 tons, resulting in a structure 22% lighter. The figure 4 shows the evolution of the fitness of the best variant as well as the average value of a population during the optimization.

![Figure 4: Evolution of the fundamental frequency](image)

The figure 5 shows the relative element thickness changes compared to the initial structure, where red means a change of +50 % and blue a change
of −50%. This shows the tendency to concentrate the mass at the bottom of the structure, and to make the top lighter.

Figure 5: Relative Thickness Modification of the Shell

4.4 Discussion

The problem optimized in this work is to be considered as an academic example and was done to test the new developed optimization method. The results cannot be directly used for the industrial applications since the stiffness is not included in the fitness function. This will be considered in future work by extending the method to multi-objective problems.

5 Weight Minimization of a Fuel Cell End Plate

In this section the performance of the developed optimization tool is shown by a second application. The weight of an end plate of a fuel cell stack is minimized.

The work was initiated by Martin Ruge who is involved in a project [7] to develop fuel cell stacks to power automotive vehicles. Figure 6 shows the general setup of such a fuel cell stack. A stack consists of 125 bipolar plates (dark brown) to transport and distribute all the different fluids. 2 end plates (brown) and 8 bolts hold the structure together and guarantee a constant pressure distribution on the bipolar plates.

5.1 Problem Description

Objective and Constraints. The weight of such an end plate is minimized subject to a stress constraint and a manufacturing constraint (plate is produced by milling). This objective is important to increase the power density (power per weight) of a fuel cell stack. In a second optimization procedure (described in [1]), the bottom surface of the weight minimized structure can independently be modified to guarantee constant pressure distribution on the fuel cells.

Discrete Design Space. A quarter model of the design space has been modeled in ANSYS, using SOLID95 elements. This is shown in Figure 7, with grey non-design elements (minimum thickness of the plate), red elements for the bolt areas and orange elements for the design space which can be modified. The coarse mesh was chosen to keep computation times in affordable limits.

Figure 6: Assembly of a Fuel Cell Stack

Figure 7: Discrete Design Space of a Quarter Model

Boundary Conditions. The end plates are loaded on the bottom surface by the specified
pressure of the fuel cells. The reaction forces, hold through the 8 bolts, are also applied as pressures on the bolt contact faces. This ensures an equal distribution of the forces on both bolts and does not restrict the rotation of the bolt areas. In addition, symmetry conditions are applied and one single node of the design space is fixed to locate the model in space.

**Material.** Due to cost considerations, the end plates are built in aluminum. This leads with a safety factor of $S_F = 1.5$, to a stress constraint of $\sigma_{\text{max}} \leq 280 \frac{N}{mm^2}$.

### 5.2 GA Coding and Fitness Function

After modeling the FE-model in ANSYS, the next step is to build the optimization model for the Genetic Algorithm. This includes the coding of the design variables and the implementation of a fitness function.

**Coding of the Design Variables.** In order to automatically introduce the manufacturing constraint, an integer string is chosen to represent the element height. This means that an integer value, which can take the values from 0 to 5, represents the thickness of each “element tower” in the design space.

For the value 0 only the grey element is active, while the value 5 represents all 6 elements over the height.

There are 325 element towers in the design space (orange). In addition the two hollow cylinders around the bolts are allowed to vary as well, adding two design variables to the set. This means that an integer string of length 327 represents the set of design variables or an individual of the GA. The number of possible solutions can then be determined with:

$$N_{\text{solutions}} = 6^{327} \approx 10^{254} \quad (9)$$

The largeness of this number explains the efforts which were put into keeping the mesh coarse and evaluating the GA in parallel!

**The Fitness Function.** Genetic Algorithms require that for each design one single value can be determined which describes the fitness of that particular solution. This fitness value $F$ is defined as a weighted combination from the objective weight minimization and the constraint on the maximal allowable stress:

$$F = c_1 \cdot V_{\text{active}} + c_2 \cdot V_{\text{stress2high}}$$

where

$$V_{\text{active}} = \text{Volume of the actual solution}$$

$$V_{\text{stress2high}} = \text{Volume of elements with } \sigma_{\text{mises}} \geq 280 \frac{N}{mm^2}$$

$c_1, c_2 = \text{Constants} \quad (10)$

The value of $c_1$ is chosen such that if $V_{\text{stress2high}} = 0$, $F$ represents the active volume in percent of the design space volume. The constant $c_2$ was adjusted that way, that a relatively small overstressed volume significantly increases the fitness value.

### 5.3 Results of Numerical Optimization

The GA optimization tool is run for 1500 generations with a population size of 100 on 17 DEC Alpha workstations. This corresponds with app. 75000 ANSYS FE-evaluations and takes about 8 h. Figure 8 shows the best individual ever found for the above configuration. The stress constraint for this design is fully observed. One can see that a rib topology has evolved, overlayed by the stochastic variations of the Genetic Algorithms.

![Figure 8: Resulting Design of GA optimization](image)
5.4 Interpretation and Final Design

Following the traditional steps of using structural optimization in product development, the “LEGO” like result has to be transformed back to CAD. The interpretation was done by Jörg Evertz from TRIBECRAFT AG and resulted in the rib-structure shown in Figures 9 and 10. The new design of the fuel cell end plate results in a weight saving of about 30% by a load increase of 80% compared to the plate which has been used up to now.

![Figure 9: Possible Interpretation of the Optimization Results by Jörg Evertz (TRIBECRAFT AG)](image)

ANSYS is integrated in this tool and is used as solver. This provides flexibility since it is so possible to optimize any objective function that ANSYS can evaluate. However, this work showed that the integration of ANSYS in other programs results in a relatively difficult handling.

In order to reduce the processing time, this tool was implemented with MPI (Message Passing Interface), which allows to evaluate the structures in a workstation-cluster.

The performance of the algorithms and of the tool is shown in two real world applications where structures are optimized for completely different objectives. The use of parallel evaluation reduces considerably the time needed for structural optimization based on genetic algorithms. However, the size of the design space is still a limiting factor for the complexity of the structure which could be optimized.

Further investigations should be done to allow multi-objective optimization of mechanical structures and to keep on reducing the time for processing the evaluation.

![Figure 10: Design of the Optimized End Plate by Jörg Evertz (TRIBECRAFT AG)](image)

6 Conclusion

This paper has presented a tool developed to optimize structures for a large range of objectives. The optimization method uses genetic algorithms which can be described as stochastic search algorithms, based on the mechanics of natural selection and natural genetics.
References


