ANNALS OF THE ORADEA UNIVERSITY
Fascicle of Management and Technological Engineering
ISSUE #1, MAY 2013, http://www.imtuoradea.ro/auo.fmte/

EVOLUTIONARY COMPUTING IN STRUCTURAL DESIGN OPTIMIZATION PROBLEMS

Razvan CAZACU[1], Lucian GRAMA[2]
[1] Industrial Engineering and Management Department, Petru Maior University, Nicolae Iorga Street, No. 1, Targu Mures, Romania, razvan.cazacu@ing.upm.ro
[2] Industrial Engineering and Management Department, Petru Maior University, Nicolae Iorga Street, No. 1, Targu Mures, Romania, lucian.grama@ing.upm.ro

***Abstract*—**Product and process optimization is a special field of interest in modern engineering. A particularly interesting subject, with great implications on the time and cost of the design and manufacturing of products is the structural design optimization. Traditionally employed by mathematical optimization procedures, modern approaches propose daring and original techniques based on natural processes and phenomena. The current paper presents the types of optimization usually employed in structural problems and an overview of the evolutionary computing paradigm, with its two main branches: evolutionary algorithms and swarm intelligence.

***Keywords*—**evolutionary computing, structural optimization, multiobjective optimization, FEA

I. INTRODUCTION

PROCESSES and products optimization has always been a main concern for engineers. Looking for better behaving, easier to obtain and cheaper products is an intrinsic aim of engineering. Structural optimization focuses on finding the best shape and size of the elements being designed in terms of material quantity, compliance, ease of manufacturing and possibly other objectives.

In structural optimization, there are three different approaches depending on the chosen parameters to be optimized: optimizing the topology, shape or size of elements. Not in all cases a clear distinction between the three can be made and their exact definition (which parameter belongs to which kind of optimization) is dependent on the problem to be solved, being different for skeletal structures (trusses and frames) and for continuum ones (plane and volumetric elements).

Fig. 1 presents visually the possibilities of optimization for a cantilever beam, considered here a plane continuous element, loaded at the free end with a load perpendicular to the beam axis. The original cantilever design space to be optimized is presented in Fig.1a. The optimization alternatives are:

1) Topology optimization (Fig.1b) – find the approximate initial layout for material distribution inside the design space;

2) Shape optimization (Fig.1c) – find the best shape by changing the control points of the parameterized contour;

3) Size optimization (Fig.1d) – describe the layout in terms of a number of parameters and find the best set of values for these parameters.

In the traditional approach, the optimization is carried out for a single type at once, keeping all the other parameters fixed. Topology is used to find an initial rough layout of the best solution inside the design space while shape and size optimization are carried out in a subsequent step to fine tune the model. However, for better results and a seamless optimization procedure there are several attempts at performing the optimization in and integrated manner, mixing in the same problem formulation topology, shape and size parameters and doing all the optimizations simultaneously [1] - [3].

From a historical point of view, the first optimization

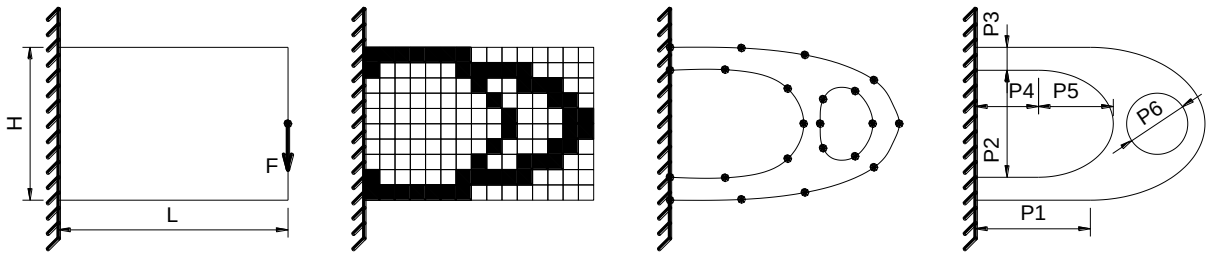

Fig.1. Structural optimization possibilities for a cantilever beam

methods were the analytical, mathematical programming ones. For the general case of nonlinear objective functions or constraints, most of the methods are gradient-based, implementing a variation of the Newton algorithm, requiring continuous, differentiable functions of real parameters.

Relatively more recent, to overcome the obvious problems classical methods were facing especially for computationally complex problems, a new family of optimization techniques was born: metaheuristic methods (from the Greek words "meta" – here meaning "extra" or "higher level" - and "heuriskein" – meaning "to find" – for a combined definition of "higher level optimum finding methods"). Metaheuristic methods generally use stochastic optimization, generating random solutions in the design space and trying to iteratively improve them with respect to the fitness function. As opposed to the gradient-based methods, metaheuristic methods:

1) Usually make no assumptions about the optimization problem, making the formulation easier and more general;
2) Can escape local solutions, being able to make jumps in the search space and find better solutions in other areas;
3) Need no information about the derivative of the function;
4) Can optimize discrete or noisy functions, not requiring them to be continuous or differentiable;
5) Scale well with the increase of dimensions; classical problems suffer from the "dimensional hell", being extremely difficult to be applied to structural optimization, typically a problem with a large number of parameters, thus dimensions;
6) Can handle very large and complex design spaces.

On the other hand, the major disadvantage of metaheuristics is that they don't guarantee a solution is ever found and have no knowledge of if and when the global solution is obtained. The iterative process is terminated after a fixed set of steps or after a solution is found within a prescribed range. At the same time, due to their stochastic nature, it's probable that different runs of the algorithms will result in different solutions. However, practice has shown metaheuristic methods work well for most optimization problems and do find optimal solutions regardless.

Metaheuristic algorithms can be divided in two major families: local search and evolutionary computing. While evolutionary computing will be detailed in the following paragraphs, local search moves iteratively towards the optimum by applying local changes to a randomly generated initial solution. Most prominent local search methods are hill-climbing and simulated annealing. Simulated annealing in particular is pretty cheap in terms of calculation cost and can also be used to search for global optima but its behaviour in performing structural optimization has been shown to be less effective than in the case of other metaheuristic methods [4]. More, its efficiency is heavily dependent on parameters choice, which in turn is problem specific and requires knowledge about the actual problem being solved. Setting the right values for the parameters becomes increasingly difficult with the dimensions, i.e. the number of problem parameters.

II. Evolutionary Computing Overview

Evolutionary computing refers to what is probably the most exotic group of optimization methods, methods generally inspired from natural phenomena or behaviors. They are said to be population-based techniques as they work simultaneously with an entire group of solutions (called a population) and evolve them iteratively towards the optimum.

As part of the metaheuristic family of optimization techniques, evolutionary computing methods implement algorithms independent of the actual problem being solved. However, knowledge about the problem can be leveraged by adapting the general optimization techniques to incorporate certain specific details. Properly handled, these adjustments can lead to significant increases in algorithm efficiency, both in terms of solution quality and computational effort and time. As a result, problem specific variations of the general optimization methods are worth pursuing in scientific research.

To emphasize this idea even more, it has been shown that the performance of all optimization methods is almost the same if averaged over all optimization problems. This principle has been formalized under the name of "no free lunch theorem" [5]. To put the theorem in other words, if a certain technique shows very good performance for a certain type of optimization problem, it will probably have a less than average performance on other problems. It is thus important to study for each type of optimization problem which algorithms are more suitable and which are not.

There are two major subfields under the evolutionary computation umbrella:

1) *Evolutionary Algorithms* (Examples: Genetic Algorithm, Genetic Programming, Memetic Algorithm, Evolutionary Programming, Evolutionary Strategy);
2) *Swarm Intelligence* (Examples: Particle Swarm Optimization, Ant Colony Optimization, Artificial Immune System).

Evolutionary Algorithms (EA) are inspired by the evolution mechanism of species. It tries to evolve a population of individuals, each a possible solution to the optimization problem, across multiple generations towards the optimal solution, imitating natural evolution concepts such as genes, fitness, selection, reproduction and mutation.

Swarm Intelligence (SI) mimics the collective behaviour of large organized systems, usually inspired from nature: flying of bird stocks, fish schooling, ants

search for food, raindrops falling, mass systems interaction, etc. As in the case of EAs, each individual in the population represents a feasible solution to the problem. Individuals are not moving in the search space (design space) in an intelligent way, but following simple principles involving local and global information. However, the sum of interactions between them and their environment gives the group as a whole intelligent behavior. The movement of the population is expected to be towards the optimum position in space (optimal solution).

The most well documented algorithms are:

1) *Genetic Algorithm* – the most representative Evolutionary Algorithm, first proposed in 1975 [6];

2) *Particle Swarm Optimization* - the most representative Swarm Intelligence method, first proposed in 1995 [7].

Not only are these algorithms the most representative for their family, but as shown in [8] they have been proven to be the most effective for structural design optimization on several benchmark problems. Some works compare the two methods as implemented in different optimization problems [4] while others suggest incorporating concepts from one technique into the other can lead to very good results.

In both EA and SI, the value of each individual is evaluated using a fitness function, the actual function to be optimized. To move the population in the direction of better fitness values and thus search for the optimum the algorithms are biased towards fitter individuals. In order to apply evolutionary computing optimization one needs procedures to encode and decode the problem and to evaluate the fitness of the encoded solutions. These are the only problem specific aspects of the methods, the ins and outs of otherwise true black-box optimization algorithms. Encoded individuals are usually represented as arrays of parameters values. Constraints can also be modeled, either as limits for the parameters or they can be evaluated from the encoded solution. In the latter case, to assure individuals not satisfying a certain constraint have less chance of influencing the general fitness of the group they have their fitness value penalized with an amount proportional to the incompliance of the individual. Typical constraints for structural design optimization are stress, displacement or vibration modes.

The optimization problem can be formalized as:

$$Min/Max(\boldsymbol{f}(\boldsymbol{x}) : \boldsymbol{X} \to \Re, \boldsymbol{X} \in \Re^{\boldsymbol{n}}) \qquad (1)$$

In the above, $\boldsymbol{f}$ is the objective function and $\boldsymbol{x}$ is an n-dimensional vector containing the design parameters, belonging to the solution space $\boldsymbol{X}$. The objective function in structural design is usually the total mass of the model, expressed in terms of design parameters. The model is subjected to constraints of stress and/or displacement. To evaluate all these Finite Element Analysis (FEA) is used and each simulation is costly in terms of computational time and effort. Population-based optimization techniques require a very large number of fitness evaluations (hundreds or even thousands), equal to the number of individuals multiplied by the number of iterations. To reduce the number of evaluations, a technique called fitness approximation [9] - [11] can be employed, leading to huge gains in algorithm running time. Fitness approximation implies building an approximate, easier to use model (also called meta-model) for the fitness function and apply it as a surrogate for the real evaluation. The approximate models use sampled data (for which an exact evaluation is needed) from the solution space and use them to find the function values for the other points. They can use polynomial interpolation, regression models, neural networks and other approximation techniques. To overcome the problems that can occur when using exclusively the approximate model, such as converging to local optima, it is advisable to use a compromise and evaluate some of the solutions with the approximate model and some of them with the exact method.

One of the best methods for structural optimization fitness approximation, a problem with an n-dimensional, discrete design space is the Adaptive Fuzzy Fitness Granulation [12]. The technique uses an adaptive group of solutions called granules, which have their fitness value calculated using exact evaluation. If a new solution is similar enough to one of these solutions, its fitness is not calculated but inherits the fitness value of the fuzzy granule it resembles. If not, the new solution is evaluated using the exact method and added to the group of granules. The adaptive nature of the method comes from the fact that the group of solutions is dynamic, receiving new solutions which are not similar to any in the group and dropping solutions for which no similarity is found for a while, but also from the fact that the granules radius of influence is adapted as needed during the run of the algorithm to allow for exploitation of good solutions.

III. Multiobjective Optimization In Evolutionary Computing

Structural optimization is often times carried out with respect to more than one objective function. This is called *Multi-objective Optimization* (MO). The process can involve the optimization of mass, compliance, manufacturability and others, under the constraints of stress, displacement or vibration modes. The constraints can also be modeled as objective functions, leaving the designer the decision of choosing for example a lighter model with bigger maximum stress or a heavier model with a bigger safety factor. The formalization of the MO is similar to single-objective optimization, but instead of the single function f, we minimize or maximize a vector of $\boldsymbol{m}$ objective functions $\boldsymbol{F(x)}$:

$$Min/Max(\boldsymbol{f}(\boldsymbol{x}) : \boldsymbol{X} \to \Re^{\boldsymbol{m}}, \boldsymbol{X} \in \Re^{\boldsymbol{n}}) \qquad (2)$$

One possible approach to solve MO is to define a super fitness function which averages the values of the

individual fitness functions, each weighted with its relative importance in the overall fitness. This super value is then used as the single measure of solution quality. One of the biggest problems with this method is that it's extremely difficult to find the right set of weighting values that can correctly reflect the total quality of a solution. More, it hides possible feasible solutions from the designer, making decisions instead of them based on rather arbitrary criteria.

To overcome these limitations, the concept of Pareto efficiency can be used. A solution is considered Pareto efficient or Pareto optimal if it's not strictly dominated by any other known solution. A solution strictly dominates another if for all fitness functions it evaluates to a better or at least equal value. Fig. 2 exemplifies a minimization problem with respect to two fitness functions f1 and f2. Solution A strictly dominates solution B, as it has a smaller value for both functions. On the same graph, the Pareto limit (or frontier) can be observed. This is the group of solutions which are Pareto efficient, not strictly dominated by any other solution in the population.

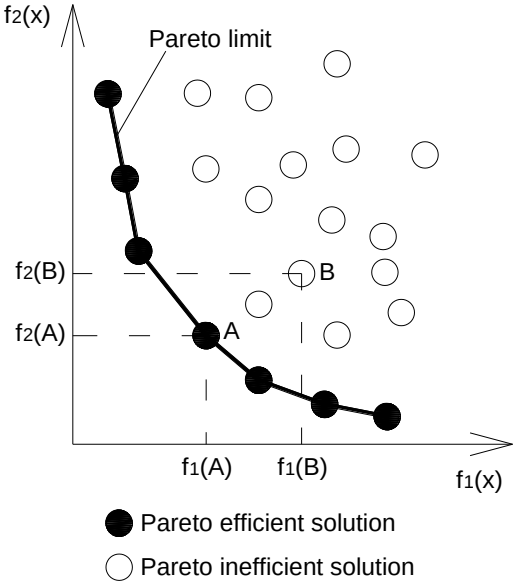

Fig.2. Pareto Efficiency for 2 objective functions

Multi-objective optimization genetic algorithm (MOGA) is the multi-objective variant of the genetic algorithm. The algorithm is more complicated than the standard single-objective one, one the biggest difficulties being setting the stopping criteria correctly. To overcome these issues, improvements to the standard MOGA have been proposed in several papers, including [13].

IV. Conclusions And Further Research

As shown by the increasing number of researchers involved in this field and the huge effort invested in refining existing and developing new methods, metaheuristic optimization has become a central part in engineering optimization. The clear advantages highlighted in this paper when compared to analytical methods recommend metaheuristics and evolutionary computing in particular as the preferred way of performing structural optimization.

Evolutionary computing works with populations of solutions in an iterative way, trying to evolve the individuals, each a candidate solution, towards the local and global minima of the objective function. The two most representative algorithms are evolutionary algorithms and particle swarm optimization. These two methods are the most studied and well-developed of all evolutionary computing algorithms. Although there is no method that can perform at maximum efficiency for all optimization problems, many papers show that the two mentioned here are generally the most effective for structural optimization. As a continuation of this study, we will detail in a future paper the inner workings and optimization possibilities offered by the genetic algorithms and particle swarm optimization.

References

[1] Fourie, P.C. and Groenwold, A.A., "The particle swarm optimization algorithm in size and shape optimization", Structural and Multidisciplinary Optimization, Volume 23, Issue 4, 2002, pp 259-267.
[2] N. Noilublao and S. Bureerat, "Simultaneous topology, shape and sizing optimisation of a three-dimensional slender truss tower using multiobjective evolutionary algorithms", Computers and Structures, Volume 89, Issues 23-24, 2011, pp 2531-2538.
[3] M. Zhou et al, "An integrated approach to topology, sizing, and shape optimization", Structural and Multidisciplinary Optimization, Volume 26, Issue 5, 2004, pp 308-317.
[4] S. Sanchez-Caballero et al, "Recent Advances in Structural Optimization", Annals of the Oradea University, Fascicle MTE, Volume XI (XXI), 2012, pp 2.118-2.127.
[5] D.H. Wolpert and W.G. Macready, "No Free Lunch Theorems for Optimization", IEEE Transactions on Evolutionary Computation, Volume 1, Issue 1, 2007, pp 67-82.
[6] J.H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, 1975, ISBN 0-262-08213-6.
[7] J. Kennedy and R. Eberhart, "Particle Swarm Optimization", Proceedings of IEEE International Conference on Neural Networks, Volume 4, 1995, pp 1942-1948.
[8] R.C. Eberhart and Y. Shi, "Comparison between genetic algorithms and particle swarm optimization", Proceedings of the 7th International Conference (EP98 San Diego), Volume 1447 of Lecture Notes in Computer Science, 1998, pp 611-616.
[9] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation", Soft Computing, Volume 9, Issue 1, 2005, pp 3-12.
[10] L. Shi and K. Rasheed, "ASAGA: an adaptive surrogate-assisted genetic algorithm", GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008, pp 1049—1056.
[11] J. Ziegler and W. Banzhaf, "Decreasing the number of evaluations in evolutionary algorithms by using a meta-model of the fitness function", Proceedings of the 6th European Conference in Genetic Programming (EuroGP'03), Volume 2610 of Lecture Notes in Computer Science, 2003, pp 264—275.
[12] M.R. Akbarzadeh-T, M. Davarynejad and M. Pariz, "Adaptive fuzzy fitness granulation for evolutionary optimization", International Journal of Approximate Reasoning, Volume 49, Issue 3, 2008, pp 523-538.
[13] S. Narayanan and S. Azarm, "On improving multiobjective genetic algorithms for design optimization", Structural Optimization, Volume 18, Issues 2-3, 1999, pp 146-155.