Object-Oriented Development of an Optimization Software in Java using Evolution Strategies

Veronika Reinauer[1], Christian Magele[2], Christian Scheiblich[1], Andrej Stermecki[2],
Remus Banucu[1], Jan Albert[1], Michael Jaindl[2], and Wolfgang M. Rucker[1]
[1] Inst. for Theory of Electr. Eng., Univ. of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
[2] Inst. for Fundamentals and Theory in Electr. Eng., Univ. of Graz, Kopernikusgasse 24/III, 8010 Graz, Austria
Email: veronika.reinauer@ite.uni-stuttgart.de

***Abstract*—Finding the optimal set of parameters of an often rather difficult system is a major task in numerical optimization. Using evolution strategies is an optimization technique based on mutation and recombination. A modular composition and thus an applicability for different problems, applying e. g. different simulation software or evaluation strategies, is provided by the use of modern software techniques like design patterns. Therefore, an innovative object-oriented software design for implementing the evolution strategies using Java is presented, discussed and proven by numerical examples.**

I. Introduction

Solving real world optimization problems has become a major task in computational electromagnetics in the recent decades. Numerous different optimization strategies, stochastic and deterministic ones as well as such looking for a single optimal solution or for many possible solutions along the Pareto optimal front have been developed and coupled in one or another way with programs solving the resulting forward problems. Recently, much effort has been put firstly on the methodology, how to develop the optimization software in a modular and hence in an expandable way and secondly on the procedure, how to call and utilize different solvers. In this paper a Java based object-oriented design of a $(\mu/\rho, \lambda)$ Evolution Strategy (ES) is presented and coupled with different solvers to compute the underlying field problems like EleFAnT2D [1] and ANSYS [2].

II. Problem Definition

The optimization software consists of two different components. The first one containing the optimization part is implemented in Java and presents the main part; see Fig. 1. Here the ES is chosen out of a set of available ones. In this work, only the $(\mu/\rho, \lambda)$ ES is described closer in detail. Additionally, the function to be minimized —called quality function— as well as its dimension are set. Furthermore, the sets of parameters are calculated in this optimization part and the feasibility of the particular sets are examined.

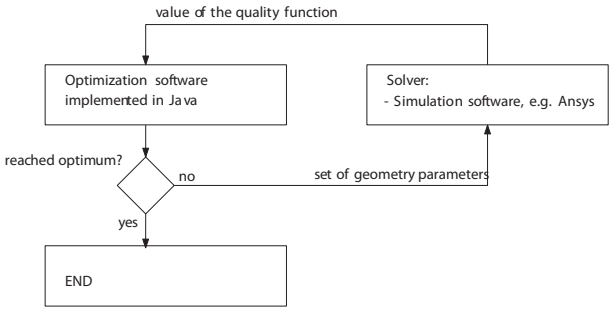

Fig. 1. Applied software and its interaction

During the procedure of optimization the values of the quality function for the calculated sets of parameters have to be determined. Regarding a magnetic field problem, the shape of a special assembly has to be optimized in order to adjust the magnetic field in a particular region. The detection of the value of the quality function —called fitness of a particular configuration— is realized using Ansys [2] or EleFAnT [1] but due to the design of the optimization software any other simulation tool can be used instead.

The value of the quality function is passed to the optimization software. The optimization is stopped, if the minimum is reached or a predefined stopping condition is fulfilled. The best set of parameters is considered to be the solution.

III. Object-Oriented Development in Java

According to the ES, an object-oriented design and implementation were established in Java. The use of design patterns [3],[4] allows for an effective computation of the parameter sets as demanded by the strategies, independent of the requested dimension of the quality function. By the object-oriented implementation the software structure of *Inheritance* is realized for the ES as well as for the *Config* objects that stores amongst other things the set of parameters as well as the corresponding value of the quality function. The superclass *Config* covers the specific subclasses, e.g. *Config2D* applied with an objective function with only two degrees of freedom; see Fig. 2.

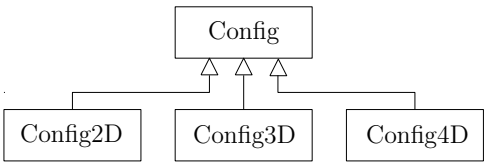

Fig. 2. UML class diagramm of the Config classes

During the optimization, the code only depends on objects of the particular superclass. Hence, the code of the optimization software is uniform. Therefore, nearly no internal maintenance is needed. For the extension of the software to another ES or to another dimension of the quality function, only a new subclass for the superclass has to be created. The rest of the code has not to be changed.

IV. $(\mu/\rho, \lambda)$ Evolution Strategy

Evolution Stategies (ES) imitate simplified biological features [5]. A higher order $(\mu/\rho, \lambda)$ ES relies on the application of population, mating and environmental selection, recombination, reproduction and mutation [6]. In the beginning an initial number of configurations is randomly generated, taking the constraints of each optimization parameter into account. Then, to move to the next generation, λ configurations (children) are generated from the μ parental ones. Firstly, a predefined number of parents (namely ρ parents) are selected for recombination from the population by taking the fitness of each parent into account (mating selection). Secondly, the chosen ones undergo what is called arithmetic cross over (1). In the $(4/2, 12)$ ES used in this paper 2 parents $\left(\mathbf{p}^i, \mathbf{p}^j\right)$ are selected out of 4 to produce 2 children $\left(\mathbf{d}^k, \mathbf{d}^{k+1}\right)$. This procedure is repeated until λ=12 children have been produced. The vectors $\mathbf{d}$ and $\mathbf{p}$ contain all trial variables to be modified and a is a number chosen randomly between $[0.8 \cdots 1]$.

$$\begin{aligned} \mathbf{d}^k_{unmut} &= a\mathbf{p}^i + (1-a)\,\mathbf{p}^j \\ \mathbf{d}^{k+1}_{unmut} &= (1-a)\,\mathbf{p}^i + a\mathbf{p}^j \end{aligned} \tag{1}$$

Each descendant inherits the step sizes from the predominant parent. The main evolutionary operator is the mutation operator. Since smaller changes happen more frequently, mutation is carried out by adding a vector $\mathbf{v}$ with normally distributed components to the parameter vector of the unmutated descendants $\mathbf{d}^k_{unmut}$ to obtain $\mathbf{d}^k_{mut}$ (2). Prior to mutation the stepwidth σ^k of each descendant is either multiplied with or divided by a factor α, which has to be determined at the beginning of the iteration process. The factor α was set to be 1.05 here.

$$\mathbf{d}^k_{mut} = \mathbf{d}^k_{unmut} + \mathbf{v}\left(0, \sigma^k(\alpha)\right) \tag{2}$$

The $(0, \sigma^k(\alpha))$ distribution makes a mutation in any arbitrary direction equally likely. Finally, μ children with the best fitness are selected to become the members of the next parental generation (environmental selection). The iterative process is stopped, when predefined criteria are reached.

V. Numerical Results

The $(\mu/\rho, \lambda)$ ES is applied to design the pole face of an electromagnet (see Fig. 3) to adjust the magnetic field in the region of interest in a prescribed way. Fig. 4 shows a zoomed view of the problem together with the four trial variables p_1 to p_4. The forward problem is solved using the finite element method package EleFAnT2D [1]. The task is to obtain a homogeneous field of $|B_z| = 0.02$ T in the region of interest, which is right in the middle of the air gap and covers 80% of the area of the pole. The optimal solution was found

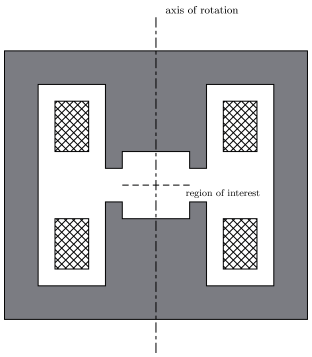

Fig. 3. Full Model of the Electromagnet.

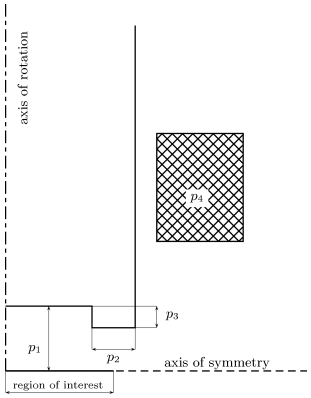

Fig. 4. Zoomed View of the Region of Interest.

after approximately 1000 iterations of the $(\mu/\rho, \lambda)$ ES. The diagrams of the initial and the optimized magnetic fields are given in Fig. 5 and Fig. 6.

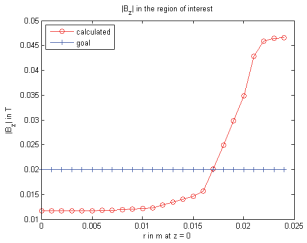

Fig. 5. Magnetic field in the region of interest, initial configuration.

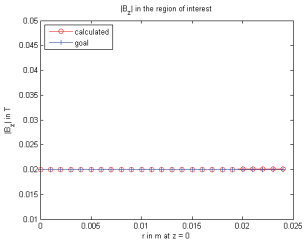

Fig. 6. Magnetic field in the region of interest, best configuration.

VI. Conclusion

This modular set-up of the software offers a high portability and is easy expandable. Due to the inheritance the optimization part with the evolution strategies has not to be modified for different requirements. The accuracy of the Java optimization software is proven by the numerical example of a pole face of an electromagnet. The optimization of the shape of an electrical motor with the proposed object oriented software coupled with Ansys will be presented in the full paper.

References

[1] Computer program package EleFAnT2D, *IGTE TUGraz*, http://www.igte.tugraz.at.

[2] Engineering Simulation Software, *Ansys Inc.*, http://www.ansys.com.

[3] S. J. Metsker, and W. C. Wake, "Design Patterns in Java," 2nd edition, *Addison-Wesley Professional*, 2006.

[4] Er. Freeman, El. Freeman, B. Bates, K. Sierra, and M. Loukides, "Head First Design Patterns," 1st edition, *O'Reilly Media*, 2004.

[5] P. Alotto, C. Eranda, B. Brandstätter, G. Fürntratt, C. Magele, G. Molinari, M. Nervi, K. Preis, M. Repetto, K. Richter, "Stochastic Algorithms in Electromagnetic Optimization," *IEEE Trans. on Magnetics*, vol. 34, no. 5, pp. 3674 - 3677, 1998.

[6] C. Magele, A. Köstinger, M. Jaindl, W. Renhart, B. Cranganu-Cretu, J. Smajic, "Niching Evolution Strategies for Simultaneously Finding Global and Pareto Optimal Solutions," *IEEE Trans. on Magnetics*, vol. 46, no. 8, pp. 2743 - 2746, 2010.