MDPI

Article

Induction motor design by utilizing AI methods

Vojtech Szekeres[1], **Ivan Zelinka**[2]

[1] VSB – Technical University of Ostrava, FEECS; vojtech.szekeres.st@vsb.cz
[2] VSB – Technical University of Ostrava, FEECS; ivan.zelinka@vsb.cz
* Correspondence: vojtech.szekeres.st@vsb.cz

Abstract: Due to increasing requirements for efficiency, price and other motor parameters it is up to the designer to find the best possible conception of the machine. Proposed paper deals with designing options of an induction machine, or any other type of electric machine that is supported by used tools, using methods of artificial intelligence (AI). Main focus is to create a convenient and simple way of design and optimization using meta-heuristic computation methods. This paper contains description of used solution, parametric model preparation, methods of evaluation with description of the cost function and used penalties, and processing of the results. With a little modification in communication script this method can be implemented to any simulation tool or program.

Keywords: artificial intelligence, optimization, differential evolution, induction motor, electric motor, motor design

1. Introduction

In recent years, manufacturers of electric machines are forced to meet ever increasing standards primarily for efficiency, but also for other properties of their products. In addition to that, the manufacturer also pushes their engineers to savings in term of cutting the cost of the motors, which puts a lot of pressure to the design of motors and challenges the contemporary design methods. Trying to squeeze out every last bit of performance per kilogram, more and more often there is no other solution than looking towards precision simulation software using finite element method or other methods.
A number of publications were issued following the quest for implementing artificial intelligence to engineering applications. Related work can be found in Evolutionary Algorithms in Engineering Applications by Dasgupta and Michalewicz [1] which in similar application used genetic adaptive search (GeneAS) to design a cylindrical pressure vessel for lowest price, Belleville spring for minimal weight with other penalties and hydrostatic thrust bearing for minimal operational power loss or even in the recently popularized article published in Nature [2] where Google engineers used deep reinforced learning method for designing the physical layout of a computer chip but also many others like [3], [4], [5], [6], [7], [8] or later [9] by the same author, but also using similar method like in this experiment in [10].
This paper proposes a simple solution for described problem using methods of artificial intelligence in cooperation with Ansys' software Rmxprt for design of electric machines. Same solution can also be implemented using finite element method (FEM) simulation for more precise design, which however requires more computational power and therefore is more time consuming.
Time is in this case very valued commodity because there are a lot of other elements, like the size of searched n-dimensional space, that noticeably affect how long it will take to get the desired results. With increasing number of searched parameters the explored space increases in its dimensions, which means the number of possible solutions rises with the power of dimensions, see Ackley's optimization test function [11] on Figure 1. These are one- and two-dimensional functions with one extra dimension as cost value of

Citation: Szekeres, V.; Zelinka, I. Induction motor design by utilizing AI methods. *Journal Not Specified* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

every solution.

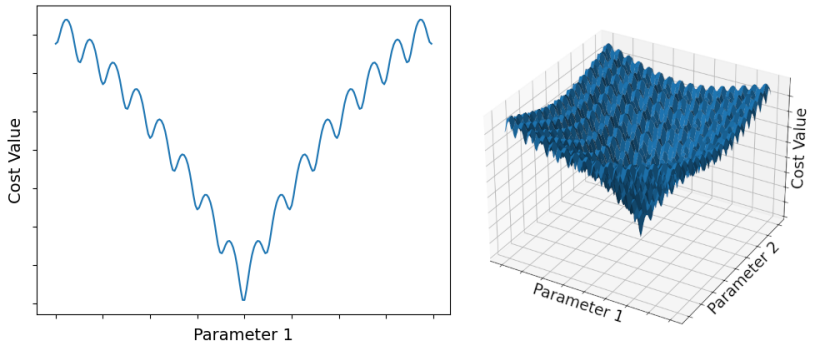

Figure 1. Representation of 1-D and 2-D functions

Figure 2 shows another example of real problems. This picture shows the space of possible solutions when synchronizing the chaotic system [12]. Even though it was a low-dimensional problem, it can be seen that the space of possible solutions is full of local extremes, and it is certainly not a problem to imagine more complex problems in multiple dimensions. From these examples, it is clear that the algorithms that are able to provide real solutions in real-time are just evolutionary algorithms and swarm intelligence.

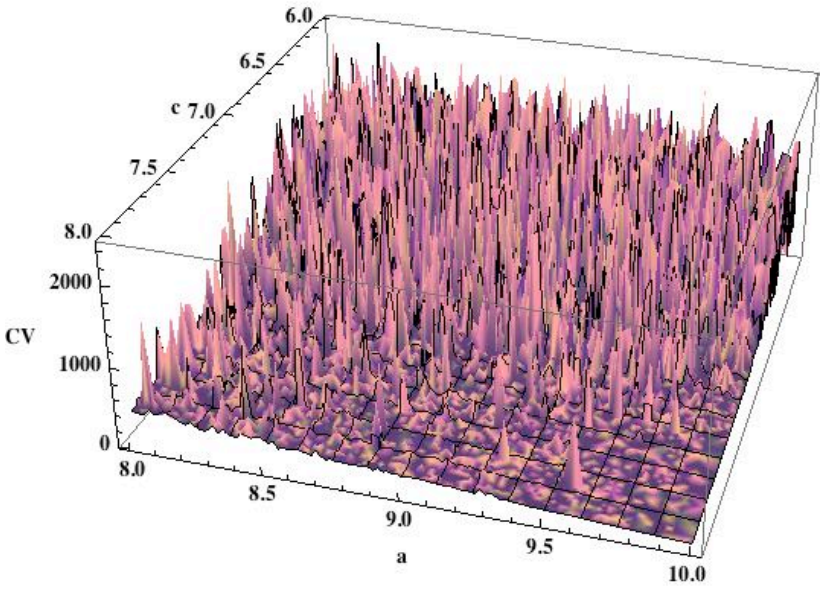

Figure 2. Real case cost function surface [12]

Therefore it is literally impossible to calculate all possible combinations in multidimensional problems and find the best solution by so-called brute force. As mentioned already in 1993 in [13] - teams of experts are starting to realize that common methods are too weak for solving highly specialized tasks. The proposed solution lies in methods using very basic artificial intelligence like particle swarm (PSO) [14], genetic algorithm (GA) [15], simulated annealing (SA) [16], self-organizing migrating algorithm (SOMA) [17], [18] or proposed differential evolution (DE) [19]. These meta-heuristic search algorithms have long been finding inspirations in natural phenomena like evolution and natural selection but also animal behavior in chase for food [20]. In this analogy, food is the desired solution of our problem. SOMA and DE algorithm constantly show the best performance among others and are very successfully applied to several various problems,

see [21]. Because their operation is well known and they are basically open sourced, researchers and enthusiasts tend to add some adjustments in certain steps [21]. That is why there is a large number of different variants of these algorithms. The one used in this paper is basic DE/rand/1, which means three random individuals are selected in mutation phase. The more detailed description of the optimization process of this algorithm has been introduced by its authors in [22] and later further investigated in [23] and [19].

2. Motivation

The evolutionary algorithms used in our experiments are known for their performance and ability to find solutions to even very complicated problems. The design of various types of equipment, including electromechanical machines, is a typical example of such use. The aim of our experiment was to verify whether these algorithms can really be used to design such devices as an induction motor and with what results. In our case, it is not a comparative study that aims to determine which algorithm is better and which is worse, but whether these algorithms can handle this type of task at all. Our experiments and their course were designed to be repeatable for anyone, with the source code available for download on the GitHub server [24].

Mentioned already in the introduction, the current design methods get challenged by requirements and standards issued by manufacturers and organizations like International Electrotechnical Commission (IEC) among others. In the past all motors were calculated using just pen and paper, meaning all the electromagnetic calculations were done by hand and the designers were obliged to have thorough knowledge of the problematic. With the advent of computers, more of the computational work could be automated which opened space for more diverse and custom designs. This is due to shorter design time but also the ability to push the performance per kilogram ratio further by designers being able to explore more variations. Nowadays the designing process works by adjusting the existing design's dimensional values and relies highly on designer's experience and knowledge in what parameters affect what characteristics and overall expertise. After finishing the dimensional design, all voltage or other variants of this motor are derived by winding changes in number of turns in coils, chording, number of parallel branches of winding. The bottleneck of this approach is the experience and level of expertise needed to design a motor effectively and overall realization of possibilities. These issues are tackled by the utilization of artificial intelligence algorithms, where all the little nuances can be implemented in evaluation process. Another great benefit of this approach is the examination of variants that otherwise would not be considered. The final design might be then reviewed by the designer and other measures may be implemented in the evaluation process that restricts unwanted parameters or their combinations. In the end, a finely tuned AI might be created with the feature of merging expertises of multiple experienced designers that is able to create cutting-edge designs in matter of hours instead of months.

3. Experiment Design

In this section, we will describe the design of our experiments, including the algorithms used, their settings of the hardware and software used and the method of evaluating the results.

The whole process of design by utilizing AI methods proposed in this paper starts by creating an initial model of the motor we want to design. This may come from an older line of motors or motor with similar parameters. The model itself is created in Rmxprt environment but the calculation method is not restricted to any concrete software and this process can be implemented to any software that allows parametrization. The searched parameters, and their boundaries in which they are going to be generated, are chosen by the designer's desire and expertise. The ones chosen for this experiment will be described in the following sections. Then the cost function code, contained in the

source code [24], must be adapted to chosen number of parameters, their names and especially their minimal and maximal value in the run-code. After this procedure, the algorithm can be started.

3.1. Used Hardware and Software

The experiment was executed on a Windows home computer setup with Intel Core i5-6600 CPU, 16GB RAM, solid-state drive, and Nvidia GTX 1080 GPU which helped in computation. The bottleneck of this setup was in used software. Ansys Electronics and its component Rmxprt was used which is relatively easy to work with and offers fast output via analytical computation. The Problem comes with a larger number of calculations with different designs, which is the main course of this experiment. With an increasing number of launched calculations, the RAM usage rises rapidly and the computational time of subsequent samples rises accordingly. This is also due to the property of Rmxprt that is saving all calculations to a dedicated folder and with a rising number of these files also rises the computational time.
One of the solutions for this might be launching every new run of the algorithm by terminating Rmxprt to free RAM, clear all the cache files that it stores for calculation results, and re-running the experiment. This may be even done in the mid-run of the algorithm after a certain number of generations.

3.2. Parametric model

As optimized model any type of motor with any chosen optimized parameter can be used. For this demonstration the designed machine is a smaller 90kW 4-pole induction motor with one layer winding. For the choice of searched parameters, four that have major impact on performance were selected.

Searched parameters are following:

- stator outer diameter (DS1)
- length of stator and rotor core (LFe)
- number of coil strands (PW)
- number of turns in one coil (N)

As mentioned earlier the experiment focuses on search for the best possible efficiency and also the power factor which helps utilizing the energy network's supply more effectively. Rmxprt itself offers option to change any value to a parameter and thus changing any dimension or amount is only a subject of changing value of certain parameter in array.

3.3. Evaluation of parameters

Every individual must go through evaluation of its parameters, i.e. either it is given a cost value via penalization or the value is calculated by defined steps which is called the **cost function**.

In this example the cost function is divided into several steps. First the algorithm sends parameters in decimals so all the numbers must be rounded to optimal format. Creating an algorithm that works with different types of numbers rather than using all floating point numbers, which then get rounded during evaluation, is a lot more complicated and can easily introduce errors into workflow. The two relatively large dimensional numbers - stator outer diameter (DS1) and length of stator (LFe) are rounded to nearest 5 due to manufacturing. That means when algorithm sends e.g. $DS1 = 228.5724$ it is rounded to value of $DS1 = 230$. Other two parameters - the number of coil strands and number of turns - are in their nature integers, so their rounded values must also be integers.

3.3.1. Penalties

Penalty function-based methods are the most well-known class of methods to handle constraints in nonlinear optimization problems. These techniques transform

the constrained problem into a sequence of unconstrained sub-problems by penalizing the objective function whenever constraints are violated. Then the goal is to force constraint violation to zero by adding a positive penalization in minimization problems or subtracting a positive penalization in maximization problems [25]. By doing this, the searched area shrinks down which helps in finding the optimal solution more efficiently. The basis for creating a penalization is in understanding of the problem and situations that might occur during optimization since the algorithm lacks the ability to do it by itself. Penalty functions implemented in this optimization problem are described in next sections.

Price

General empirical assumption in assignment like this is the largest possible machine with stator slot leaking copper, figuratively speaking. This solution being naturally cost-ineffective, a penalization must be introduced which sets constraints in price of the machine.

```
threshold = 100
if price < threshold:
    penalty = 0
else:
    penalty = f(price,threshold)
```

To emphasize the approximate price we want to get, the penalization function is recalculated via empirically obtained pseudo-exponential function (3.1) with minimum at the desired price and values lower than that equal to 0.

$$f(price, threshold) = 1.15^{(price-threshold)^2} - 1 \tag{3.1}$$

The price itself is estimated by calculating the price of one layer of stator sheet with copper wires. This number is then multiplied by number of needed sheets for given length of stator core. The influence of winding overhang is neglected by this calculation. This is representation of soft penalty which doesn't break the cost evaluation, just degrades the individuals that distance themselves from desired price.

Fill factor

Due to technological limitations the stator slot fill factor must also be constrained by highest possible value. Empirically it is set to value of 85%. This penalty is implemented by simple if-statement.

```
if ff > 85:
return 100.0
```

The fill factor (ff) is calculated as follows:

$$ff = \frac{PW \cdot S \cdot N}{A} \tag{3.2}$$

, where S is cross section of one wire and A is net area of one stator slot.

To save a lot of unnecessary computation, the fill factor is calculated before the parameters are sent to Ansys for evaluation. If it is less than the maximum value, it then continues to standard evaluation process. Unlike price penalty, the fill factor is a hard class penalty that breaks cost evaluation process and returns a very high fixed value resulting in rejection of given solution.

3.3.2. Cost function

Individuals, i.e. sets of parameters that are eligible for cost evaluation are automatically sent to simulation software as values for induction motor model parameters.

These model parameters are pre-defined during the creation of the model. It is then calculated for efficiency and power factor that is eventually evaluated by cost function f. The whole process is show on Figure 3 with *penalized* branch being a hard penalty.

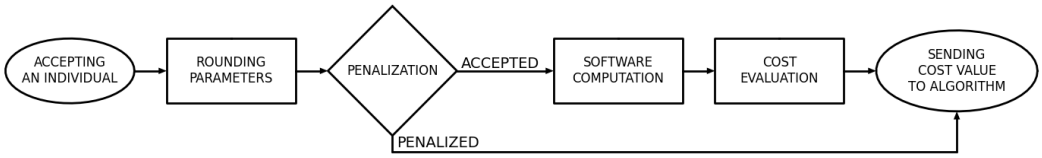

Figure 3. Flowchart of cost evaluation process

The cost function is often represented as a mathematical expression and its minimalization or maximalization results into finding the optimal solution to a given problem. Because the cost value of the optimal solution is unknown, a cost function is defined so the minimal or maximal value is idealized. That means the cost value of evaluated solutions never exceeds or even reaches 0. In mathematical notation:

$$f(x) \geq f(x_0) \tag{3.3}$$

, where $f(x)$ is the cost value of current solution and $f(x_0)$ the idealized value.

In case of our search for the optimal solution, the cost function is defined with minimum value in 100% efficiency and power factor of 1. Because the algorithm is set to minimalization, both the efficiency and power factor are formulated in $(1 - x)$ form. To combine multiple objectives into a single value a weighting factor is typically introduced. Then the overall cost value is calculated as a sum of individual results:

$$f_o = \Sigma w_i o_i \tag{3.4}$$

, where w_i are weighting factors and o_i cost values of every objective.

According to [1] there are several problems using this approach, like combination of widely different characteristics into a single weighted number or subjective choice of weighting factors. In summary - this method is rather subjective and limited in its ability to express the designer's true intent. Nevertheless, despite these difficulties, this method is frequently used. To settle the value of efficiency in cost equation a weighting factor is introduced and set to empirical value of 3.

The cost function is then defined as:

$$f = 3 \cdot (1 - \eta) + (1 - cos\varphi) + p \tag{3.5}$$

, where p equals price penalty from Equation 3.1, η is efficiency in range $[0, 1]$ and $cos\varphi$ is the power factor.

Not mentioned before, the price equation (3.1) is already designed to be on the same level as other agents of cost function in range close to threshold so there is no need to incorporate another weighting factor.

3.4. Optimization algorithm

As mentioned in the introduction the algorithm used in this demonstration is called differential evolution, introduced by its authors in 1997 [19]. Since then, it got a myriad of derivations by other authors. The one utilized here is basic scheme DE/rand/1, named by Storn [23] or DE1 in original technical report [22], with a slight variation in handling parameters in noise vector that escape the defined search space. Modification lies in the method of returning those missed parameters. With original approach these parameters are either discarded or put back in random position within pre-set boundaries. Solution used here is returning them randomly to the area 0 to 20% from the edge they "slipped"

from, see Figure 4. This ensures that the solution will keep its direction and won't be discarded.

Differential evolution has three control parameters for setting up its working conditions and behavior. Size of population (NP), crossover probability (CR) and a factor that controls the amplification of the differential variation [23] or a mutation factor (F) with one terminating parameter - number of generations (G). Limiting the number of generations i.e. using the number of generations as a terminating parameter is not needed in case other terminating conditions are introduced like a convergence test. The one used in this demonstration works on the principle of comparing best solutions in consecutive generations. If no change in best solution occurs during 10 consecutive generations, the algorithm terminates itself. In other means a parameter for number of consecutive generations with unchanging solution might be applied instead of number of generations.

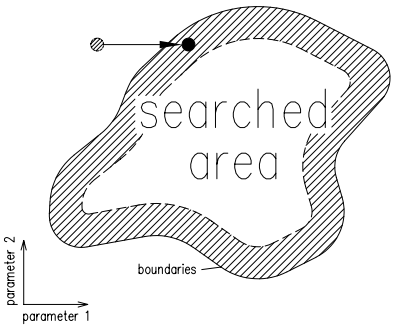

Figure 4. Graphical representation of returning slipped parameters

Since its invention, DE's has been tested extensively against artificial and real-world minimization problems [23]. Thanks to that the individual control parameters have some recommended values summarized in [23] and set accordingly in this demonstration. The number of generations (G) is set to be high enough to get to the optimal solution. Convergence test being implemented, this number doesn't play a big role in overall performance of the algorithm.

The used setup is following:

```
NP = 40
F = 0.8
CR = 0.9
G = 50
```

4. Results

To ensure the correctness of obtained solution, the optimization process ran five times with the same setup.

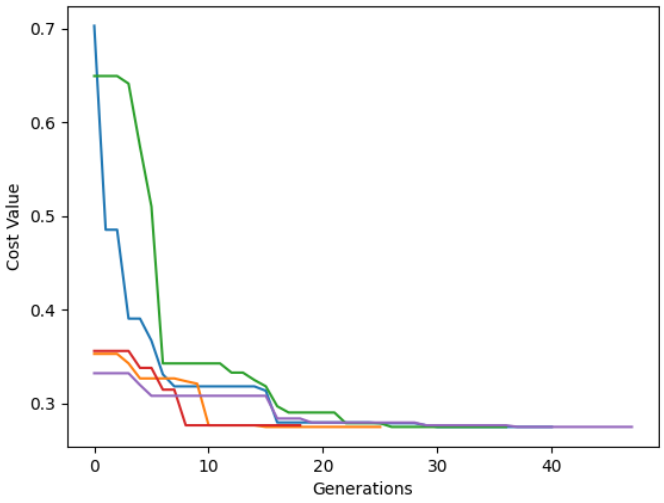

Figure 5. Course of best individuals throughout generations

Due to unknown problems with Rmxprt in running a large number of consecutive calculations, described in Section 3.1 the whole process took several hours. Overcoming this problem might help in lowering the computational time resulting in possibility of increasing the number of process launches and therefore legitimizing the results. Nevertheless, the summary of best individual in every generation, see Figure 5, shows every run, distinguished by color, resulting to the same position. Every run of optimization process reached the same value of the cost function from Equation 3.5 of ≈ 0.2748 which confirms the achieved result.
As mentioned in Section 3.3.2, the desired value in a cost function is idealized and therefore the cost value of a solution never reaches zero. It just touches this level by a slight variation. The difference between the idealized value and real result is purely based on the design of the cost function and thus the relatively low number has no scientific meaning outside this experiment, meaning the cost values of two similar experiments cannot be analyzed or compared together if the cost function is designed differently. That is why the number 0.2748 mentioned earlier has no actual value other than it is the lowest possible number found by our experiment in conditions defined and described in Section 3.3.

The motor parameters of our solution are as follows:

stator outer diameter $DS1 = 435mm$
length of stator and rotor core $LFe = 300mm$
number of coil strands $PW = 10$
number of turns in one coil $N = 16$

Resulting efficiency and power factor are:

$\eta = 95.411\%$
$cos\varphi = 0.8627$

These values correspond to efficiency class IE3 according to international standard [26]. Achieved results, but also related theoretic knowledge, imply that there are other areas for investigation to get to even higher efficiency class which may be subject of another utilization of this method.

5. Conclusion

A successful demonstration of designing certain parameters of induction motor was presented. Our experiment proofs that evolutionary algorithms can be successfully utilized also in induction, or in that matter any other, motor design. Due to nature of meta-heuristic algorithms, we can never imply that this might be the best solution possible but it is "good enough" to satisfy our conditions. Using these methods of artificial intelligence, it is true that the solution is as good as the information that is provided. Better solutions might be found with more sophisticated models, better simulation software and especially more comprehensive definitions of the problem like penalties and other constrictions. Another worth mentioning might be comparing this method with multiple individual searches and gradually implementing their results into model.
Using the proposed tools, the presented approach can be implemented to any chosen parameter of a model in Rmxprt to find best value for any electrical specification. In a relatively extreme situation, it can be used to design a whole machine all at once. However this would require a lot of other penalizations, better definition of cost equation and last but not least a lot of computational power and time. In comparison, this would be like trying to calculate the presented design just by brute force.
As mentioned in Section 3.1, a lot of computational power was wasted by inefficient operation of Rmxprt during multiple consecutive calculations. This plays a huge role in optimization of this method. Unfortunately, Rmxprt is a closed-source software and therefore the only option is to change its behavior in its settings, if possible. Another option is to use different software for motor calculation, sadly unknown to us.

To get a more precise solution, an option of creating the initial design using Rmxprt with wider range of optimized parameters and then more precise FEM computation in narrower area of these parameters is offered.

Author Contributions: Conceptualization, Szekeres V.; methodology, Szekeres V.; software, Szekeres V.; validation, Szekeres V. and Zelinka I.; formal analysis, Szekeres V. and Zelinka I.; investigation, Szekeres V.; resources, Szekeres V.; data curation, Szekeres V.; writing—original draft preparation, Szekeres V.; writing—review and editing, Szekeres V. and Zelinka I.; visualization, Szekeres V. and Zelinka I.; supervision, Zelinka I.; project administration, Szekeres V.. All authors have read and agreed to the published version of the manuscript.

Funding: This article was supported by SP2021/84 "Výzkum v oblasti zemních proudových polí, asynchronních motorů v provozních podmínkách, zpětné vlivy regulovaných pohonů, tvarová optimalizace elektrických strojů.". The authors acknowledge for support.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
DE	Differential Evolution
FEM	Finite Element Method
GA	Genetic Algorithm
GeneAS	Genetic Adaptive Search
IEC	International Electrotechnical Commission
PSO	Particle Swarm Optimization
SA	Simulated Annealing
SOMA	Self-Organising Migrating Algorithm

References

1. Dasgupta, D.; Michalewicz, Z. *Evolutionary Algorithms in Engineering Applications*; Springer Berlin Heidelberg: Berlin, Heidelberg, 1997. doi:10.1007/978-3-662-03423-1.
2. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.W.; Songhori, E.; Wang, S.; Lee, Y.J.; Johnson, E.; Pathak, O.; Nazi, A.; Pak, J.; Tong, A.; Srinivasa, K.; Hang, W.; Tuncer, E.; Le, Q.V.; Laudon, J.; Ho, R.; Carpenter, R.; Dean, J. A graph placement methodology for fast chip design. *Nature* **2021-06-10**, *594*, 207–212. doi:10.1038/s41586-021-03544-w.
3. Patel, J.L.; Rana, P.B.; Lalwani, D.I. Optimization of mechanical design problems using advanced optimization technique. *IOP Conference Series: Materials Science and Engineering* **2018-12-01**, *455*. doi:10.1088/1757-899X/455/1/012091.
4. Sandgren, E. Nonlinear Integer and Discrete Programming in Mechanical Design Optimization. *Journal of Mechanical Design* **1990-06-01**, *112*, 223–229. doi:10.1115/1.2912596.
5. Zhang, C.; Wang, H.P.B. Mixed-Discrete Nonlinear Optimization with Simulated Annealing. *Engineering Optimization* **1993**, *21*, 277–291. doi:10.1080/03052159308940980.
6. Das, A.; Pratihar, D. Optimal design of machine elements using a genetic algorithm **2002**. *83*, 97–104.
7. Chen, J.; Tsao, Y. Optimal design of machine elements using genetic algorithms. *Journal of the Chinese Society of Mechanical Engineers, Transactions of the Chinese Institute of Engineers, Series C/Chung-Kuo Chi Hsueh Kung Ch'eng Hsuebo Pao* **1993**, *14*, 193–199.
8. Cao, Y.; Wu, Q. Mechanical design optimization by mixed-variable evolutionary programming. *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC* **1997**, pp. 443–446.
9. Cao, Y.; Jiang, L.; Wu, Q. An evolutionary programming approach to mixed-variable optimization problems. *Applied Mathematical Modelling* **2000**, *24*, 931–942. doi:10.1016/S0307-904X(00)00026-3.
10. Lampinen, J.; Zelinka, I. Mechanical Engineering Design Optimization by Differential Evolution. In *New Ideas in Optimization*, 1 ed.; McGraw-Hill: London, 1999; p. 20.
11. Ackley, D.H. *A connectionist machine for genetic hillclimbing*; Kluwer Academic Publishers: Boston, 1987.
12. Zelinka, I.; Celikovskỳ, S.; Richter, H.; Chen, G. *Evolutionary algorithms and chaotic systems*; Vol. 267, Springer, 2010.

13. Mařík, V.; Štěpánková, O.; Lažanský, J. *Artificial intelligence*, 1 ed.; Academia: Prague, 1993-.
14. Kennedy, J.; Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*; IEEE, 1995; pp. 1942–1948. doi:10.1109/ICNN.1995.488968.
15. Holland, J.H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, 1975.
16. Kirkpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983-05-13**, *220*, 671–680. doi:10.1126/science.220.4598.671.
17. Zelinka, I. SOMA—self-organizing migrating algorithm. In *New optimization techniques in engineering*; Springer, 2004; pp. 167–217.
18. Zelinka, I. SOMA—self-organizing migrating algorithm. In *Self-Organizing Migrating Algorithm*; Springer, 2016; pp. 3–49.
19. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* **1997**, *11*, 341–359. doi:10.1023/A:1008202821328.
20. Dréo, J. *Metaheuristics for hard optimization*; Springer: Berlin, 2006.
21. Zelinka, I. *Artificial intelligence*, 1. vyd ed.; BEN - technical literature: Prague, 2002.
22. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report tr-95-012, International Computer Science Institute, Berkeley, 1995.
23. Storn, R. On the usage of differential evolution for function optimization **1996**. pp. 519–523. doi:10.1109/NAFIPS.1996.534789.
24. Vojtech, S. Induction Motor Design by Utilizing AI Methods, 2021.
25. Zelinka, I.; Snášel, V.; Abraham, A. *Handbook of optimization*; Springer: New York, 2013.
26. *IEC 60034-30-1*. Geneva, Switzerland, 12.0 ed., 2010.