TEKA. COMMISSION OF MOTORIZATION AND ENERGETICS IN AGRICULTURE – 2017, Vol. 17, No.1, 5-14

Implementation of genetic algorithm in electric machines optimization using Netbeans IDE and Java

V. Pliuhin[1], M. Sukhonos[1], A. Petrenko[1], A. Ehorov[2]

[1]*O.M. Beketov National University of Urban Economy in Kharkiv; e-mail: vlad.plyugin@gmail.com*
[2]*National Technical University «Kharkiv Polytechnic Institute»; e-mail: toe@mail.ru*

Received January 10. 2017: accepted January 27. 2017

Summary. In this paper the comparative analysis of existent optimization algorithms efficiency is made. The substantive provisions of genetic algorithms theory are considered. Possibility of applicability of genetic algorithms in the optimal design of electric machines was investigated. Results over of the classic genetic algorithm practical realization in induction motors optimization with a squirrel-cage rotor are done.

Key words: design, induction motor, optimization, objective function, genetic algorithm, efficiency, criteria, program.

INTRODUCTION

Neuron networks, being one of perspective directions of researches in area of artificial intelligence, were created as a result of watching processes, what be going on in the human nervous system. By the same way genetic algorithms were also «invented» and watched yet not the human nervous system but the process of living organisms' evolution.

Genetic algorithms – one of researches directions in the area of artificial intelligence, engaging in creation of living organism's evolution simplified models for the solving of optimization tasks [1–4].

RESEARCHES PROBLEM

Now for the solving of optimization tasks different methods are used, which in general case it is possible to classify on continuous, discrete and integer. In turn, the transferred tasks are divided into integer, unidimensional and multidimensional [5].

The basic problem of the applied optimization algorithms is a search of function extremum in cases with a nonlinear search area and development of methods on reduction of search time and computer resources in intricate problems [6].

Using of genetic algorithm (GA) in optimization allows at the minimal time and calculable resources to get the extremum of objective function, is examined in this paper. The aim of work is consideration of features of classic GA application in electric machines optimal design and implementation efficiency criteria search.

CLASSIC GENETIC ALGORITHM STRUCTURE

At description GA use determinations, adopted from genetics. For instance, speaking about the population of individuals, as base concepts a gene, chromosome, genotype, phenotype (Fig. 1) are used [8, 9].

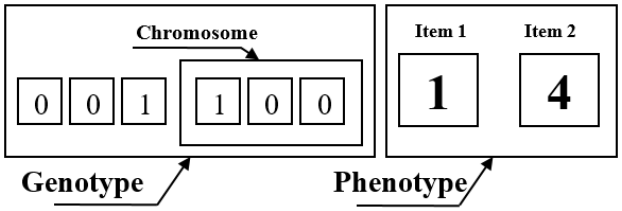

Fig. 1. Genotype and phenotype in GA

In given gene example (Fig. 1) the number «001» represents the first chromosome in binary notation and corresponds to «1» in decimal notation for «Item 1». The second chromosome with number «100» in binary notation corresponds to «4» in decimal notation for «Item 2».

Some determinations of GA theory correspond to terms from a technical vocabulary, in particular, «circuit», «binary sequence», «structure» [7].

Classic GA consists of next steps (Fig. 2):

1) initializing, or choice of initial chromosomes population;

2) an estimation of adjusted chromosomes in a population – calculation of adjusted function for every chromosome;

3) verification of algorithm stop condition;

4) chromosomes selection – selection of those chromosomes which will take part in creation of descendants for a next population;

5) application of genetic operators is mutations and crossing;

6) forming of new population;

7) selection of the «best» chromosome.

Simple GA generates an initial population of casual character. Genetic algorithm work is an iteration process which proceeds until the set number of generations or some another stop criteria will not be executed. On every generation a proportional selection will be realized on adjusting, crossing and mutation. Chromosomes, got as a result of application of genetic operators to the chromosomes of temporal paternal population, are included in the complement of new population. They become so-called current population [10] for this iteration of GA (Fig. 3).

To every chromosome, designated ch_i for i = 1, 2…N (where N is a quantity of population) corresponds

the sector of a wheel $v(ch_i)$, represented in percent format according to equations (1), (2):

$$v(ch_i) = p_s(ch_i) \times 100, \quad (1)$$

$$p_s(ch_i) = \frac{F(ch_i)}{\sum_{i=1}^{N} F(ch_i)}, \quad (2)$$

where: $F(ch_i)$ – a value of function of adjusted of chromosome of ch_i;

$p_s(ch_i)$ – chance of ch_i chromosome selection.

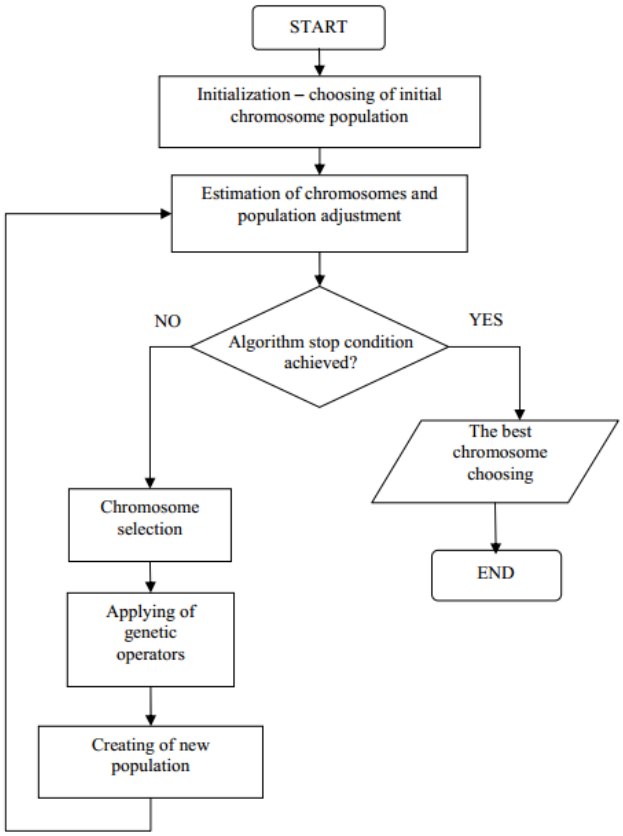

Fig. 2. The classic GA structure

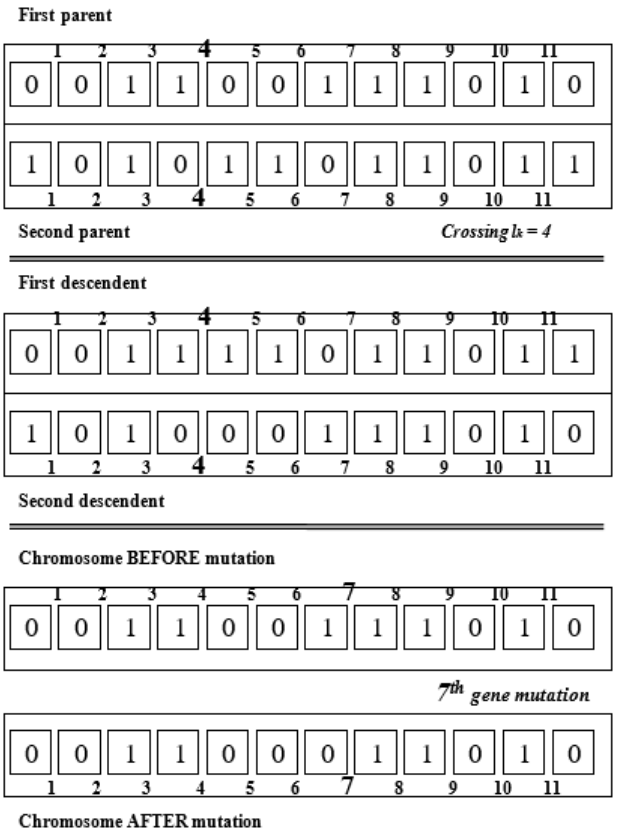

Fig. 3. Example of crossing realization in GA

The chromosome selection can be presented as a result of roulette's wheel turn (Fig. 4) as a «winning» (i.e. chosen) chromosome behaves to the falling out sector of this wheel [7].

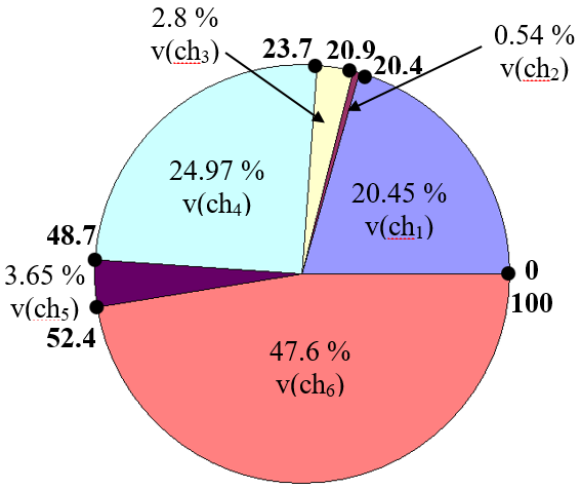

Fig. 4. Roulette's wheel in GA

Obviously, that the bigger sector, the high chance of «victory» of corresponding chromosome. Therefore, probability of current chromosome choice is proportional to the value of it adjusting function.

If we represent roulette wheel sectors as a digital interval [0, 100], the chromosome choice can be evaluated with the number choice from the interval [A, B], where A and B designate beginning and completion of circumference fragment, corresponding to some sector of wheel, where $0 \le A < B \le 100$. In this case a choice by means of roulette wheel is taken to the choice of number from an interval [0, 100], which corresponds to the concrete point on the circumference of wheel. There are also other methods of selection [8, 9].

The crossing operation consists in a chain fragments exchange the between two paternal chromosomes. The pair of parents for crossing get out from a paternal pool casual character so that probability of choice of concrete chromosome for crossing was equal to probability p_c. For example, if we have parents with two chosen chromosomes from a paternal population with quantity N, then $p_c = 2/N$. Analogically, if from a paternal population with quantity N a $2z$ chromosomes ($z < N/2$), which form z parent's pairs, were chosen, then $p_c = 2z/N$. We will pay attention, that if all chromosomes of current population

are incorporated in pair to crossing, then $p_c = 1$.

After the crossing operation parents in a paternal population substituted by their descendants.

The operation of mutation changes the values of genes in chromosomes with the probability p_m. It results in inverting of values of the selected genes from 0 to 1 and back. Value p_m, as a rule, very small, therefore the mutation of small amount of genes is exposed. Crossing is a key operator of GA and determine their possibilities and efficiency. A mutation plays more limited role. It enters in a population some variety and warns losses which would happen because of exception of some meaningful gene as a result of crossing.

As a result of selection process a paternal population, also called a paternal pool, is created with the quantity *N*, equal to a quantity of current population.

The optimization order of the GA [6] differs from previously considered for Cartesian Product [11, 12, 13] and have the next steps:

1) setting the range of the varied variables;

2) setting limitations;

3) choosing the optimality criteria;

4) calling the GA optimization function and getting the optimal set of the varied variables;

5) for founding set calling the function of automatic aim object (in our case – induction motor) calculation.

On every next iteration the values of adjusted function of settle accounts for all chromosomes of this population are calculated. The stop algorithm condition is whereupon checked up and a result is either fixed as a chromosome with the high value of adjusted function or comes true passing to the next step of GA, i.e. to the selection.

GA PROGRAMMATIC REALIZATION

We will consider an example of induction motor with squirrel cage rotor (IM) optimization with programmatic GA realization on Java in the freeware IDE NetBeans. For a decision of the set problem we will use freely expandable Java library EvoJ (“Evolution Java”) [14]. A project EvoJ is planned as upgradable framework of Java classes for the solving of various optimization tasks by means of evolutional (genetic) algorithms. For the use of EvoJ a programmer has to implement one simple interface, consisting of one method only. All other steps undertake EvoJ algorithm.

In an example it will be considered two varied variables: internal diameter of stator core and length of stator core [15, 16].

We create Java interface with the name *Solution*, in which we set the turn-down (minimum and maximal values) of the varied variables.

```
//Code of Solution interface:
package MotorClasses;
import
net.sourceforge.evoj.core.annotation.MutationRange;
import net.sourceforge.evoj.core.annotation.Range;

public interface Solution {
   //Stator core inner diameter limits
   String smin1 = "165";
   String smax1 = "205";
   //Stator core length limits
   String smin2 = "115";
   String smax2 = "145";

   //@MutationRange("0.1")
   @Range(min = smin1, max = smax1)
   double getX(); //return an optimal diameter

   @Range(min = smin2, max = smax2)
   double getY(); //return an optimal length
}
```

EvoJ is able to change the variable without a range changes of variables [14]. However, if it is needed to implement own mutation strategy, we need to declare setters because in other case variables cannot be changed.

The Evoj library does not allow to change a kind, range and step of the varied variables during dynamic implementation of the computer program, foreseeing their change only in the interface code of the computer program. Due to the authors’ applied modifications this fault was removed.

Now a range and step of the varied variables of GA allow to change directly during implementation of the program on a computer. In addition, modified GA is adapted for application in electrical machines object-oriented design.

Pay attention to annotation *@Range*, that sets the range of values which a variable can accept. Variables are initiated by casual values from the set range. However, as a result of mutation, they potentially can go out for the indicated range. It can be prevented, using the parameter *strict=«true»*, that will not allow a variable to take on an impermissible value, even if to make an effort to propose him, using setter-method.

Another moment on which it is needed to pay attention here, it that all parameters of all code annotations in EvoJ are lines, it allows, both to specify the value of parameter directly and specify the name *property* instead of concrete value, hardly not to specify the value of annotation parameters in *compile-time.*

We have an interface with variables, now we will write a fitness-function. It is recommended to implement this interface indirectly, through helper-classes which undertake some service functions: elimination of old decisions (if the maximal decision life time is set), cashing of function value for decisions which were not sifted from on the previous GA iteration.

A fitness-function for our case will look like the next (we create a new class with the name *Rating*):

```
//Fitness-function
package MotorClasses;
import net. source for ge. evoj. strategies. sorting.
Abstract Simple Rating;

public class Rating extends AbstractSimpleRating
<Solution> {
   static AMotor mot;//object of motor
   static int krit;//index of optimality criterion
```

```
   static int iter_numb;//number of iterations

 //Designer
   public void set_motor(AMotor mot, int krit){
      this.mot = mot;
      this.krit = krit;
      this.iter_numb = 0;
   }
   //get iterations number
   public int get_iter(){return this.iter_numb;};

   public static double calcFunction(Solution solution){
      iter_numb++;//increase number of iterations
      double x = solution.getX();//get new diameter
      double y = solution.getY();//get new length
      mot.stator.set_D(x/1000);//setting of new diameter
      mot.stator.set_ld(y/1000);//setting of new length
     //automatic motor parameters calculation
      double fn = mot.auto(krit);

      return fn;//return optimality criterion
   }

   @Override
   public Comparable doCalcRating(Solution solution){
      //call of calculation function
      double fn = calcFunction(solution);
      boolean flag = mot.control();//control of limitations

      if (Double.isNaN(fn) | flag == false){
         return null;//screening-out of false variant
      } else {
         return - fn;//return of effective variant
      }
   }
}
```

Here all obviously enough, we simply take and count our function, using variables from the *Solution* interface. Because we search minimum, and the contract of class supposes that the best decisions are due to have the greater rating, then we return the value of function, increased on –1.

In addition, we sift from false decisions (if NaN turned out or limitations were not passed), returning *null.*

In the IM class *Motor* we create the function of automatic calculation, which differs in that accepts as an argument the index of optimality criterion and returns the got criterion after the motor calculation:

```
double auto(int khit){
int res = 0;
//Body of motor calculation code
//…
switch (krit){
         case 1://1 is Efficiency
            res = 1/kpdnr;
            break;
         case 2://2 – Power Factor
            res = 1/cosFinr;
            break;
         case 3://3 – Start Current Ratio
            res = I1pn;
            break;
        case 4://4 – Start Torque Ratio
            res = 1/Mpo;
            break;
}
    return res;//return of criterion depending on it index
}
```

Further in the IM class *Motor* we create the function of GA realization (a structure is explained in comments below):

```
void optimization(int krit){
DefaultPoolFactory pf = new DefaultPoolFactory();
//creation of populations
GenePool<Solution> pool = pf.createPool(populations,
Solution.class, null);
Rating rtg = new Rating();//designer of class of Rating
rtg.set_motor(this, krit);//setting of optimum criteria
//factory of primary solutions generation
DefaultHandler handler = new DefaultHandler(rtg, null,
null, null);

//making of iterations in populations
handler.iterate(pool, iterations);
//get the best found solution
Solution solution = pool.getBestSolution();

D_opt = solution.getX()/1000; //optimal diameter
L_opt = solution.getY()/1000; //optimal length
int iter = rtg.get_iter(); //get the number of iterations

}
```

The code of GA optimization starting consists only of two lines:

```
//motor.limits();//setting of limitations
//motor.optimization(krit);//optimization with the criteria
krit
```

If a solution does not satisfy, it is possible to continue the iterations of GA (increasing the number of populations and iterations), while the desired quality of solution will not be attained [15]. Example of GA realization in listing 1 is shown.

Listing 1. Example of GA realization

```
//GA is the main module
public void GeneAlgorithm(int var, int krit){
DefaultPoolFactory pf = new DefaultPoolFactory();
GenePool<Solution> pool = pf.createPool(populations,
Solution.class, null);
Rating rtg = new Rating();
rtg.set_motor(this, krit);
DefaultHandler handler = new DefaultHandler(rtg, null,
null, null);
handler.iterate(pool, iterations);
Solution solution = pool.getBestSolution();
double x = solution.getX();
double y = solution.getY();
int iter = rtg.get_iter();
opt_pareto = new Vector();
```

```
opt_pareto.add(x/1000);
opt_pareto.add(y/1000);
opt_pareto.add(iter);
  } //End of the main module of HA

//GA main realization
public class Rating extends AbstractSimpleRating
<Solution> {
static Reactor reactor;
static int krit;
static int iter_numb;
static boolean flag;

public void set_reactor(Reactor react, int krit){
this.reactor = react;
this.krit = krit;
this.iter_numb = 0;
this.flag = false;
 }
 public int get_iter(){return this.iter_numb;};

 public static double calcFunction(Solution solution){
       flag = true;
       iter_numb++;
       double Di = (double) solution.getInnerDiameter();
       double Dw = (double) solution.getWireDiameter();
       int nt = solution.getTurns();
       int np = solution.getNumParallelWires();
       reactor.set_innerDiameter((double)Di);
       ((RoundReactor) reactor).set_diameter(Dw/1000);
       reactor.set_numTurns(nt);
       reactor.set_numParallelWire(np);
       ((RoundReactor) reactor).ControlGA();//wire
       double fn =
((RoundReactor)reactor).CalculationGA(krit);
       //Last control
       flag = reactor.Control();
        return fn;
    }

    @Override
    public Comparable doCalcRating(Solution solution){
double fn = calcFunction(solution);
       if (Double.isNaN(fn) | flag == false){
          return null;
       } else {
          return - fn;
       }
    }
}

//GA interface of the varied variables
public interface Solution {
    //Inner Diameter
    String smin1 = "20";
    String smax1 = "1000";
    //Wire Diameter (x1000)
    String smin2 = "50";
    String smax2 = "4000";
    //Number of Turns
    String smin3 = "1";
    String smax3 = "1000";
    //Number of Parallel Wires
    String smin4 = "1";
    String smax4 = "3";

    @MutationRange("10")
    @Range(min = smin1, max = smax1, strict = "true")
    int getInnerDiameter();

    @MutationRange("1")
    @Range(min = smin2, max = smax2, strict = "true")
    int getWireDiameter();

    @MutationRange("1")
    @Range(min = smin3, max = smax3, strict = "true")
    int getTurns();

    @MutationRange("1")
    @Range(min = smin4, max = smax4, strict = "true")
    int getNumParallelWires();

}
```

The algorithm of selection of effective variants is based on the Pareto preference rule [5]. According to this rule, from the array of acceptable variants selected a variant *Ko*, from *Ko* = 1, and for all *j* criteria a condition show below is checking up:

$$F_{kj} < F_{koj}, \quad k = 1, 3; \quad j = 1, 3. \tag{3}$$

Variants, dissatisfying to this condition (3), are cast aside as scienter "bad", because yield to other on all criteria. A new variant gets out from other variants and got an index *Ko*. Condition (3) is checked up again. A process recurs until there will be not a single variant which *Ko* index would not be appropriated. Remaining variants will make the array of effective variants.

The construction of effective variants array allows considerably to narrow a search area, but the problem of optimal variant selection remains [9].

At the small number of effective variants selection of the best from them comes true on the basis of careful analysis of every variant taking into account the requirements of technology factors, standardization, unification and other factors, which are not taken into account in a model.

If the number of effective variants is great, then often use convolving of criteria. We will use one of convolving methods.

Let F_j^* – the record value of *j*-criteria among effective variants, and F_{kj} is a value of *j*-criteria in *k*–variant. Then size:

$$W_{kj} = \frac{F_{kj} - F^*_j}{F^*_j} \tag{4}$$

determines as far as concrete variant worse than record one by a *j*-factor. We will designate the size of W_{kj} for a worst variant as W^*_j and will execute the rate fixing as below:

$$\hat{W}_{kj} = W_{kj} / W^*_j . \quad (5)$$

If to enter the gravimetric coefficients ξ_j for every criterion, then it is possible to form the generalized criterion [8]:

$$F_\xi = \frac{1}{3}\sum_{j=1}^{3} \xi_j \cdot \hat{W}_{kj} \to \min.$$

A variant having the least value F_ξ is most near to the record and, consequently, is the best (optimal) at set vector ξ of relative meaningfulness of criteria. Changing the elements of vector ξ in accordance with one or another preferences, it is possible to get the different best variants.

In the educational planning formal comparison of three variants is executed by means of the generalized criterion of F_ξ supposing identical meaningfulness of all private criteria ($\xi_1 = \xi_2 = \xi_3 = 1$). Results of calculations are added to the Table 1.

Table 1. Form for selection of optimal variant

Variant index	W_1	W_2	W_3	F_ξ
1				
2				
…				

A variant having the most record indexes in a Table 1 by one or two parameters W_i is the best for selection. The selected variant is considered as an optimal. Selection Java-code of the best candidate in listing 2 is shown.

Listing 2. Selection of the best candidate by Pareto

```
public static int Pareto(Vector eff, int[] krit){
    int Nopt = 0;//index of optimal variant
    //vector of optimal solutions (index and selection of
Fw Pareto)
    opt_pareto = new Vector();
    int Ni = eff.size();//number of effective variants
    int Nj = krit.length;//number of gravimetric
coefficients
    //comparative values of current variant with a record
    double[][] W = new double[Ni][Nj];
    double[][] W1 = new double[Ni][Nj];//array of the
rate fixing of W
    double[] Wmax = new double[Nj];//array of worst
values of W
    double[] Fmax = new double[Nj];//array of record
indexes
    double[] Fw = new double[Ni];//generalized criterion
of optimality
    double[] Fwp = new double[Ni];//generalized criterion
of optimality in %

    for (int i = 0; i < Nj;i++){
      Fmax[i] = Double.POSITIVE_INFINITY;
    }
    //Search of record indexes
    for (int i = 0; i < Ni; ++i){
      double[] temp = new double[Nj];
      temp = (double[]) eff.get(i);
      for (int j = 0; j < Nj; ++j){
        if (temp[j] < Fmax[j]){
          Fmax[j] = temp[j];
        }
      }
    }

    //Calculation of comparative indexes
    for (int i = 0; i < Ni; ++i){
      double[] temp = new double[Nj];
      temp = (double[]) eff.get(i);
      for (int j = 0; j < Nj; ++j){
        W[i][j] = (temp[j] - Fmax[j])/Fmax[j];
      }
    }

//A search of worst value is in the array of W
//A worst value is maximal divergence with a record
    for (int i = 0; i < Nj; i++){
      Wmax[i] = W[0][i];
    }
    for (int i = 1; i < Ni; ++i){
      for (int j = 0; j < Nj; ++j){
        if (W[i][j] > W[i - 1][j]){Wmax[j] = W[i][j];}
      }
    }

//Rate fixing
 //A worst variant will have a greater value W1 and equal
to 1.0
    for (int i = 0; i < Ni; ++i){
      for (int j = 0; j < Nj; ++j){
        W1[i][j] = W[i][j]/Wmax[j];
      }
    }
    //Calculation of the generalized optimality criterion
    //The best index will have a less value Fw
    //A current index gets (diminishes) better a
gravimetric coefficient
    //Range of gravimetric coefficient:
    //from 1 (a correction is not present) to 100 (max.
correction)
    for (int i = 0; i < Ni; ++i){
      for (int j = 0; j < Nj; j++){
        Fw[i] += W1[i][j]/((double)krit[j]);
      }
      Fw[i]/=Nj;
    }
    //Selection of the best variant
    double Fpmin = Double.POSITIVE_INFINITY;
    double Fpmax = Double.NEGATIVE_INFINITY;
    for (int i = 1; i < Ni; ++i){
      if (Fw[i] < Fpmin){
        Fpmin = Fw[i];
        Nopt = i;
      }
      if (Fw[i] > Fpmax){Fpmax = Fw[i];}
    }
    //Return of resulting array Fw in %
    for (int i = 0; i < Ni; ++i){
      if (Fw[i] == Fpmin){
```

```
            Fwp[i] = 100 - (Fw[i] - Fpmin)*100/Fpmax;
          }
          else{
            Fwp[i] = 100 - Fw[i]*100/Fpmax;
          }
        }
    opt_pareto.add(Fwp);//array of Pareto function Fw in %
      opt_pareto.add(Nopt);//number of optimal variant
     //Return of optimal variant index
      return Nopt;
}
```

So for implementation of GA using EvoJ library it is necessary:

1) to create an interface with variables;

2) to implement the interface of fitness function;

3) to create the population of solutions and carry out the necessary amount of iterations of GA above them, using a code, given above.

OPTIMIZATION RESULTS

As an object of optimal design an induction electric motor with squirrel cage rotor AIR 4 kW, 380 V, 50 Hz, 750 rpm of base industrial execution was taken (Fig. 5).

Fig. 5. Induction motor of AIR series

For design project an induction motor with squirrel cage rotor [15–18] with an object-oriented class structure, grounded before in was worked out [19–23]. In accordance with a class structure was drawn up an object-oriented UML diagram [24–28].

A project was extended by the database of insulation materials, copper wires and nomenclature of electrical grade steels.

Methodology of induction motor object-oriented design was realized as a software product in IDE NetBeans on Java, allowing to entry of data in program windows, represented calculation results as tables and scaled charts.

The got design results of base machine [24, 29, 30] in next project were taken as basic data for induction motor optimization.

As the varied parameters core length and stator core inner diameter were chosen. As an optimality optimization criterion maximal efficiency parameter was set.

Results of GA implementation with maximum efficiency parameter as an optimality criterion in a Table 2 is shown.

As it obvious from a Table 2, efficiency is higher in an optimal motor and other parameters not over than permissible limits.

Table 2. Genetic algorithm: motor parameters before and after optimization

Name	Base value	Optimal value
Air gap flux density, T	0.748	0.807
Stator core inner diameter D, mm	185	194
Stator core length L_δ, mm	130	115
Relative size $\lambda = L_\delta / \tau$ ($\tau = \pi D/2p$), where p – number of pole pairs	0.895	0.755
Stator slot height, mm	21.9	14.6
Rotor slot height, mm	32.2	33.2
Width of stator slot top line, mm	7.7	7.8
Width of stator slot bottom, mm	10.2	9.3
Rotor slot upper diameter, mm	7.9	7.8
Rotor slot bottom diameter, mm	3.7	3.4
Efficiency	0.825	0.891
Power factor	0.893	0.90
Starting current ratio	5.84	6.52
Starting torque ratio	1.4	1.62
Maximum torque ration	2.65	2.88
Overall stator winding temperature, Cel.	93.3	95.7

What kind of optimization algorithm to choose – the designer has to decide. If importance of getting of optimal result outweighs expenses on its receipt, then one often ignore in course of calculation time, and it is possible to apply the "heavy" optimization algorithm (Cartesian product (CP) as an instance).

When it is needed to produce evaluation calculations in the maximally compressed time, and quality of the got results is written into a permissible error, then it is possible to use fast-acting, but less accurate algorithms (GA with global and single optimization criteria).

CONCLUSIONS

1. Algorithm of the previous considered CP [11] in comparison with the GA, allows to execute multi-criterion optimization, that is its undoubted advantage. In addition, the CP always gives only the synonymous best variant among the existing ones. However, in the range of varying of two variables ± 20 % from a base value (3976 combinations) the calculation time is approached up to 48 min.

2. Implementation of CA gives stunning results. At the same varied variables and range of their change ± 100 % from a base value, the calculation time is only 40 sec. However, GA, at least, in the present paper task, does not allow to execute optimization for a few criteria.

3. In the GA number of the varied variables and a range of their change is not important from the point of view of the productivity, because a set of the varied variables is created dynamically, but not beforehand, as in the CP method. In addition, all combinations of the variables and values of objective function are realized in a binary form. However, time of the GA work is very critical to the number of the created populations and

number of iterations in the populations.

4. The selection of population's number and iterations realized by an experienced way increases until an acceptable result will not be obtained. A result of the optimization with the use of the GA will always be the best for the chosen criterion, but there is not a guarantee, that a better variant can exist. The GA productivity at effective variant populations number is determined but not by the number and range of the varied variables.

5. When production of approximate calculations in maximum compressed terms is needed and quality of the obtained results is written with a permissible error, then using of the GA optimization will be the irreplaceable instrument for designers

REFERENCES

1. **Cochehurova E.A. 2012.** Teorija i metody optimizacii [Theory and optimization methods], Tomsk: Tomsk polytechnic institute, 2012, 157. (in Russian).
2. **Goldberg D. 1989.** Genetic algorithms in search, optimization and machine learning, Boston: Addison-Wesley, 1989, 412. (in English).
3. **Poli R., Langdon W., McPhee N.** A field guide to genetic programming, http://lulu.com, 2008, 250. (in English).
4. **Copin B**. **2004.** Artificial intelligence illuminated, USA, 2004, 739. (in English).
5. **Reklejtis G. 1986.** Optimizacija v tehnike [Optimization in technics], Moscow, Mir, 1986, 351. (in Russian).
6. **Zablodskij N., Pliugin V., Lettl J., Buhr K., Khomitskiy S. 2013**. Induction Motor Design by Use of Genetic Optimization Algorithms, Prague, Transactions on electrical engineering, 2013, No. 3, 65 – 69. (in English).
7. **Emel'janov V.V. 2003.** Teorija i praktika jevoljucionnogo modelirovanija [Theory and practice of evolution modelling], Moscow, Phizmatlit, 2003, 432. (in Russian).
8. **Zablodskij N., Pliugin V., Buhr K. 2013**. SAPR elektromehanichnih pristroïv [CAD of electromechanical devices], Alchevsk, Lado, 2013, Part 2, 320. (in Ukrainian).
9. **Polivyanchuk A. 2016.** Application of multi-criteria analysis in evaluating the efficiency of diesel particulate filters / TEKA. COMMISSION OF MOTORIZATION AND ENERGETICS IN AGRICULTURE. – Vol. 16, No.3, 15-20. (in English).
10. **Palis S., Leidhold R., Pliuhin V., Maslennikov A. 2015.** Primenenie geneticheskogo algoritma optimizacii v proektirovanii jelektricheskih mashin [Using of genetic algorithm in electric machines design], Kharkiv, NTU "KhPI", 2015, 303 – 306. (in Russian)
11. **Zablodskij N., Pliugin V., Lettl J., Buhr K. 2013.** Induction Motor Optimal Design by Use of Cartesian Product, Prague, Transactions on electrical engineering, 2013. No. 2, 54 - 58. (in English).
12. **Immrich W., Klavzar S., Rall D. 2008.** Topics in graph theory: graphs and their Cartesian product, Taylor & Francis, 2008, 205. (in English).
13. **Daepp U., Gorkin P.** Reading, writing and proving: a closer look at mathematics, Springer, 2011, 395. (in English).
14. **Inprise Corporation. 2002.** EvoJ - Evolutionary computations framework, Inprise Corporation, http://evoj-frmw.appspot.com, 2002. (in English).
15. **Pliuhin V. 2014.** Teoreticheskie osnovy ob'ektno-orientirovannogo rascheta i proektirovanija jelektromehanicheskih ustrojstv [Theoretical basis of object-oriented calculation and design of electromechanical devices], Alchevsk, Lado, 2014, 200. (in Russian).
16. **Pyrhonen J., Jokinen T., Hrabovcova V. 2013.** Design of rotating electrical machines, John Willey & Sons, 2013, 616. (in English).
17. **Pliugin V., Milykh V., Polivianchuk A., Zablodskij N. 2015.** Using of object-oriented design principles in mathematic modeling of electric machines, Lublin-Rzeszow, TECA, Vol. 15, No. 2., 2015, 25 – 32. (in English).
18. **Tessarolo A. 2015.** Modeling and analysis of multiphase electric machines for high power applications, Springer, 2015, 270. (in English).
19. **Zablodskii N.N., Pliugin V.E., Petrenko A.N. 2016.** Using object-oriented design principles in electric machines development, Kharkiv, Electrical Engineering & Electromechanics, 2016, No. 1, 17–20. (in English).
20. **Shalloway A., Trott J. 2004.** Design patterns explained: a new perspective on object-oriented design, Pearson education, 2004, 480. (in English).
21. **Unhelkar B. 2005.** Practical object oriented design, Cengage learning Australia, 2005, 236. (in English).
22. **Robinson P. 1992.** Hierarchical object oriented design, Prentice hall international (UK), 1992, 238. (in English).
23. **Gilbert S., McCarty B. 1998.** Object-oriented design in Java, Waite group press, 1998, 731. (in English).
24. **Pliugin V., Shilkova L., Letl J., Buhr K., Fajtl R. 2015.** Analysis of the Electromagnetic Field of Electric Machines Based on Object-oriented Design Principles, Prague, PIERS 2015, 2015, 2522 - 2527. (in English).
25. **Page-Jones M. 2002.** Fundamentals of object-oriented design in UML, Addison-Wesley, 2002, 288. (in English).
26. **Barclay K., Savage J. 2003.** Object-oriented design with UML and Java, Butterworth-Heinemann, 2003, 428. (in English).
27. **Hunt J. 2013.** The unified process for practitioners: object oriented design, UML and Java, Springer, 2013, 281. (in English).
28. **Levrik E., Havdal V. 2002.** Java the UML way: integrating object-oriented design and programming, Willey, 2002, 754. (in English).
29. **Zablodskiy N., Pliugin V. 2015.** 3D magnetic field distribution in a screw double-stator induction motor, Lviv, CPEE 2015, 2015. 239 – 241. (in English).
30. **Zablodskii N.N., Plyugin V.E., Gritsyuk V.Y., Grin' G.M. 2016.** Polyfunctional electromechanical energy transformers for technological purposes, Russian Electrical Engineering, 2016, No. 87 (3), 140. (in English).

РЕАЛИЗАЦИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА В ОПТИМИЗАЦИИ ЭЛЕКТРИЧЕСКИХ МАШИН С ИСПОЛЬЗОВАНИЕМ NETBEANS IDE И JAVA.

Плюгин В.Е, Сухонос М.К., Петренко А.Н., Егоров А.В.

Аннотация. В данной статье выполнен сравнительный анализ эффективности существующих алгоритмов оптимизации. Рассмотрены основные положения теории генетических алгоритмов. Исследована возможность применимости генетических алгоритмов в оптимальном проектировании электрических машин. В качестве метода оптимизации электрических машин используются генетические алгоритмы, а в качестве объектно-ориентированного языка используется язык Java с библиотекой EvoJ, в котором переменные класса интерфейса генетических алгоритмов задаются динамически во время выполнения программы. Разработано программное обеспечение с оконным интерфейсом в среде Netbeans IDE. Приведены результаты практической реализации классического генетического алгоритма приоптимизации асинхронного двигателя с короткозамкнутым ротором.

Ключевые слова: проектирование, электрическая машина, оптимизация, целевая функция, генетический алгоритм, КПД, критерий, программа.