Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Signal Generation for Switched Reluctance Motors using Parallel Genetic Algorithms

Mike Eichhorn * **Sandro Purfürst** ** **Yuri A.W. Shardt** *

* *Department of Automation Engineering, Technical University of Ilmenau, Helmholtzplatz 5, 98693 Ilmenau, Germany (e-mail:(mike.eichhorn, yuri.shardt)@tu-ilmenau.de)*
** *NIDEC driveXpert GmbH, Ehrenbergstraße 11, 98693 Ilmenau Germany (e-mail:spu@drivexpert.de)*

Abstract: Switched reluctance motors (SRM) are an inherent part in robotics and automation systems where energy and cost efficiency is required. This motor type has no windings and permanent magnets on the rotor which results in a simple and robust structure. However, SRMs require a complex electronic control system to generate a specified number of voltage pulses for each motor phase. This paper presents the signal generation of multiple phases using only one current sensor in an asymmetric half bridge (AHB). In addition to maintain the predetermined phase voltages, sufficient current measurement windows and a minimal current ripple for the individual phases are further optimization criteria for signal generation. The generation of a state vector which controls the individual semiconductor for each motor phase to achieve a required phase voltage and simultaneously fulfill the multi-objective optimization criteria is challenging. Due to the vast number of possible solutions, a genetic algorithm (GA) was used to find state combinations that are suitable for the formulated optimization criteria. The results were discussed and recommendations about the genotype representation and the used genetic operators were given. Interested readers will find detailed information about the software technical implementation using the Global Optimization Toolbox from MATLAB.

Keywords: Genetic algorithms, Parallel programs, Switched reluctance motors, Measuring span, Multi-objective optimization

1. INTRODUCTION

The working principle of switched reluctance motors (SRMs) has been known for more than 100 years. However, research on the topic has increased only during the past three decades. Due to its simple and low cost construction without permanent magnets, SRMs are highly reliable and can reach mechanical velocities of up to more than $100{\times}10^3$ min^{-1}. This motor type can also be used for position control and since it does not require brushes, this motor can be used in hazardous and explosive environments.
Many researchers have tried to find optimal design parameters with the help of techniques inspired by natural evolution such as Shaked and Rabinovici (2006); Sihem et al. (2012); Prabhu et al. (2015). These results are the motivation for this paper to focus on the optimization of switching signals for semiconductors. For cost reduction, integration density improvement and redundancy, several position sensorless algorithms have been developed. Using one current sensor for multiple measurements nearly simultaneously is another concept discussed in this paper. The focus here is on an asymmetric half bridge converter topology with one current sensor for two phases, based on Kjaer and Gallegos-Lopez (1997) and Chen and Lu (2013). Fig. 1 a) shows the asymmetric half bridge for a two phases with possible current sensor locations. Fig. 1 b) shows a variant, where only one current sensor is used to sense all phase currents. This requires a single current path through the sensor for an unambiguous measurement of each phase current. Thus, the objectives of this paper are: (i) to present a circuit state-based way of looking at switched reluctance motor control and (ii) to propose methods to generate a control sequence using genetic algorithms (GA) with multi-objective fitness functions.

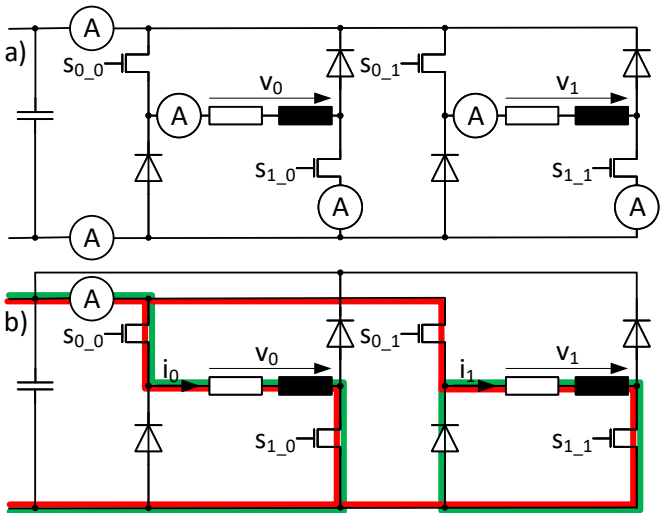

Fig. 1. AHB with a) different current sensor placements and b) a single sensor. The lines show current paths for a valid (green) and an invalid (red) measurement.

2. GENETIC ALGORITHM

2.1 Representation

Fig. 1 shows an asymmetric half bridge with two switches $n_s = 2$ for each of the two phases $n_p = 2$. This results in 16 possible states z_i in one time step T_i. A control sequence for the SRM with n_{st} time steps can be described easily with a state vector $\bar{Z}$ consisting of n_{st} states. This state vector, which can be called phenotype in this form, has to be transformed into a suitable genotype representation to be operated by the genetic algorithm. First of all, one individual can be understood as an array containing the switching states s_{s_p} of each semiconductor. This way a binary string represents the genetic information for a specific individual which is shown in Fig. 2.
The decimal value h_{st} of an asymmetric half bridge with n_s switches for each of the n_p phases is defined using

$$h_{st} = \sum_{j=0}^{n_p-1} \sum_{i=0}^{n_s-1} (s_{i\_j} 2^{n_s n_p - j n_s - i - 1}) \tag{1}$$

which is used as the state number for the practical implementation.

The switching state of every semiconductor can be considered as a single gene with two alleles. The first allele stands for a conducting switch (=1) and the second allele stands for non-conducting switch (=0). More detailed information about the state vector concept can be found in Purfürst (2019).

The following three different types of genotype representation are discussed in this paper:

Binary-representation
Every bit of the whole stream can be manipulated independently. The genetic algorithm only operates at the binary level.

Grouped-binary-representation
The genetic algorithm is not allowed to apply the crossover mechanism to each bit of the stream. Only specific points between binary groups of $n_s n_p$ bits are possible breakages. Instead, the mutation operator can be used on every bit.

Integer-representation
The whole bit stream is interpreted as a number of binary groups represented by their integer value h_{st}. Genetic operators only work at this level.

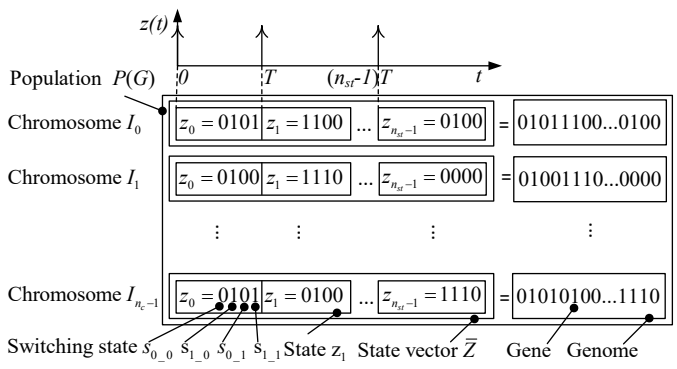

Fig. 2. Population of bit string individuals for a SRM with $n_s = 2$ and $n_p = 2$ according Fig. 1

2.2 Fitness function formulation

The use of a state vector for control of an asymmetric half bridge will have some effects on the electrical, and furthermore, the magnetic and mechanical behavior of the motor. The resulting CPU requirements also have to be taken into account. The following list contains possible properties:

- average phase voltages
- current ripples
- current measurement windows
- capacitors balancing
- switching frequency
- switching equality between high-side and low-side switches
- necessary memory space
- efficiency

In this paper, the first three requirements will be considered.

To define a fitness function that evaluates the fitness of each individual corresponding to the phase voltage achievement, an error needs to be defined that represents the deviation between a number of specified reference phase voltages $v_{p_{ref}}$ and the actual average voltages $\overline{v_{p_{act}}(\bar{Z})}$ forced by the state vector for each phase. The fitness value $fit_{voltage}$ can be calculated using the Euclidean distance from the error vector $|\mathbf{e}|$ as follows:

$$fit_{voltage} = |\mathbf{e}| = \sqrt{\begin{matrix}(v_{0_{ref}} - \overline{v_{0_{act}}(\bar{Z})})^2 + ...+ \\ (v_{n_p-1_{ref}} - \overline{v_{n_p-1_{act}}(\bar{Z})})^2\end{matrix}} \tag{2}$$

For the provision of current measurement windows, the genetic algorithm is used to minimize the difference between the desired number of measurement windows $n_{p_{m\_des}}$ and the actual number of measurement windows $n_{p_{m\_act}}$ for a specified phase current i_p in phase p. The corresponding fitness function fit_{window_p} is therefore

$$fit_{window_p} = max(n_{p_{m\_des}} - n_{p_{m\_act}}, 0) \tag{3}$$

The max operator allows the user to set the lower bound of the fitness value to zero, because additional measurement windows are not necessary after achieving the desired number $n_{p_{m\_des}}$.

The ripple which represents the difference between the maximum current $max(i_p(\bar{Z}))$ and the minimum current $min(i_p(\bar{Z}))$ of a phase controlled by the state vector is the third optimization criterion for this paper which can be calculated as

$$fit_{ripple_p} = max(i_p(\bar{Z})) - min(i_p(\bar{Z})) \tag{4}$$

To merge all fitness functions described above, a weighting function needs to be defined. This over-all fitness function allows to weight each optimization criterion using the weighting factors $w_{voltage}$, w_{ripple_p} and w_{window_p}, that is,

$$\begin{aligned} fit_\Sigma = & w_{voltage} fit_{voltage} + \\ & w_{ripple_0} fit_{ripple_0} + ..+ \\ & w_{ripple_{n_p-1}} fit_{ripple_{n_p-1}} + \\ & w_{window_0} fit_{window_0} + ..+ \\ & w_{window_{n_p-1}} fit_{window_{n_p-1}} \end{aligned} \tag{5}$$

3. PRACTICAL IMPLEMENTATION

For the practical implementation a set of properties for all possible states needs to be defined. The system to be implemented is represented by $2^{n_s n_p} = 16$ states which is shown in Table 1. Each of these states forces a phase voltage v_p which results in a current i_p. To be able to measure the phase currents with a reduced number of phase current sensors, every state includes information about the measurability of the necessary phase currents (green marked fields in Table 1). Each state is accessed by a decimal number index h_{st} from (1) represented by a binary code that includes information about the switching state s_{s_p} of each semiconductor.

Table 1. State properties

State Number h_{st}	Binary Code	Voltage v_0	Voltage v_1	Meas. $p = 0$	Meas. $p = 1$
0	0000	-12 V	-12 V	0	0
1	0001	-12 V	0 V	0	0
2	0010	-12 V	0 V	0	1
3	0011	-12 V	+12 V	0	1
4	0100	0 V	-12 V	0	0
5	0101	0 V	0 V	0	0
6	0110	0 V	0 V	0	1
7	0111	0 V	+12 V	0	1
8	1000	0 V	-12 V	1	0
9	1001	0 V	0 V	1	0
10	1010	0 V	0 V	0	0
11	1011	0 V	+12 V	0	0
12	1100	+12 V	-12 V	1	0
13	1101	+12 V	0 V	1	0
14	1110	+12 V	0 V	0	0
15	1111	+12 V	+12 V	0	0

3.1 Program details

For the implementation of the genetic algorithms, MATLAB and Simulink were used. The Global Optimization Toolbox (MathWorks (2020a)) from MathWorks provides methods to solve problems with multiple optima. It includes genetic algorithms which have a wide range of standard mechanisms. The toolbox allows the creation of a custom genetic algorithm by modifying the functions for population initialization, fitness scaling, parent selection, crossover, and mutation. The multi-objective genetic algorithm can also solve multiple-objective optimization problems by identifying the Pareto front. The possible simultaneous calculation of the fitness function for the several individuals enables parallelization using of the Parallel Computing Toolbox (MathWorks (2020b)). This gives a rapid speedup during the optimization of the GA in case of a computationally intensive fitness function. For a realistic and fast simulation of the electrical behaviour of the asymmetric half bridge within a Simulink environment, the PLECS Blockset (Plexim (2020)) was used. Fig. 3 shows the resulting Simulink-system with the integrated PLECS Circuit block.
The GA in MATLAB uses the standard interface for optimization routines, and therefore, allows easy access. A good example for the usage of a binary genetic algorithm in MATLAB is described in (Babatunde (2020); Babatunde et al. (2014)). Listing 1 shows a possible implementation using a grouped-binary-representation.

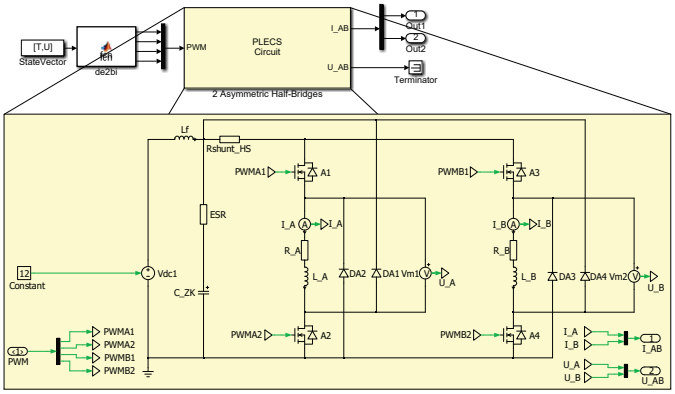

Fig. 3. Simulink system of the two simultaneously excited asymmetric half bridges using the PLECS Blockset

Listing 1. MATLAB code example

```
% Define the GA parameters
options = gaoptimset(
'PopulationSize',200,...
'Generations',150,...
'SelectionFcn',{@selectionroulette},...
'CrossoverFraction',0.7,...
'FitnessScalingFcn',{@fitscalingrank},...
'EliteCount',4,...
'StallGenLimit',100,...
'PlotFcns',{@gaplotbestf},...
'Display','iter',...
'CreationFcn',{@gacreationuniform},...
'PopulationType','bitstring',...
'MutationFcn',{@mutationuniform,0.02},...
'CrossoverFcn',{@crossovertwopointByte},...
'UseParallel',true);
% Define the quality requirements
weightVec =
[wVoltage wWindow0 wWindow1 wRipple0 wRipple1];
refVoltageVec = [10 10];
measurementWindowVec = [5 5];
% Define the fitness function handle
fitnessFcn = {@fitnessFcnSRM,refVoltageVec,
measurementWindowVec,weightVec};
% Start the GA
[bestChromosome,bestQ,~,~,population,scores] =
ga(fitnessFcn,nVars,options);
```

4. RESULTS

The results presented in this section are based on the GA parameter settings in Table 2. The basics section in the upper part of the table includes parameters that are used in each representation (see section 2.1). A weighted fitness function according to (5) was used for the requirements evaluation of the several individuals. All weighting factors for the used requirements ($w_{voltage}$, w_{ripple_p} and w_{window_p}) were set to 1.0. This pragmatic weighting factor combination leads to desired results. The settings in the following table are dependent on the individual representations. The parameter mutation fraction was adapted to achieve a maximum convergence rate for the several representations. For most functions in the GA's, the standard MATLAB procedures are used. However, for the crossover function in the grouped-binary-representation and the creation and mutation function of the integer-representation, custom functions had to be designed.

Table 2. GA parameters

GA parameter	value
Population size	200
Number of generations	150
Crossover fraction	0.7
Fitness function	Rank based fitness scaling
Elite count	4
Mutation fraction	0.005 (Bin.), 0.02 (Group.), 0.04 (Int.)
Binary and grouped-binary-representation	
Creation function	Uniform
Population type	Bitstring
Mutation function	Uniform
Binary and integer-representation	
Crossover	Two point crossover
Grouped-binary-representation	
Crossover	Two point crossover (custom function)
Integer-representation	
Creation function	Uniform integer (custom function)
Population type	Double vector
Mutation function	Uniform integer (custom function)

4.1 Convergence behavior

The three representations presented in Section 2.1 were used for convergence analysis. Each phase should have an average voltage value of about 10 V. As can be seen in Fig. 4 b), the best individual is not changed anymore after generation 54. Fig. 4 a) and 4 c) will need more generations to get the solution. The reason for the worse convergence of the binary-representation in comparison to the grouped-binary-representation is due to "additional mutations" when a crossover is carried out as a result of the arbitrary choice of breakages. This often leads to new states if the break position is not an integer multiply of $n_s \times n_p$. The reason for the poorer convergence of the integer-representation in comparison to the grouped-binary-representation needs to be searched for in their mutation algorithm. In the first step, the mutation algorithm selects several states in the state vector of an individual. Secondly, the algorithm replaces these state elements with a uniformly distributed random number between 0 and 15 ($2^{n_s n_p}-1$). For comparison, the mutation algorithm used in the binary- or grouped-binary-representation changes only by single bits in the bit string. This leads to mutated states which only differ in one bit from the original state. This optimal mutation behavior cannot be reproduced by the integer mutation algorithm. Furthermore, the mutation fraction parameter of the integer-representation, where the genotype length corresponds to the state vector length of 50, needs to be increased with regards to the grouped-binary-representation. For these reasons, the grouped-binary-representation is favored and will be used for the following optimizations.

Fig. 5 shows the convergence of the GA. Each row contains the states of the best state vector found in each generation. The color indicates the state number which is generated by the switching states of each semiconductor as a binary string converted into its decimal representation. It shows that the evolutionary convergence is very fast until generation 60. After that, only small changes can be observed until the algorithm finds the optimum at generation 120.

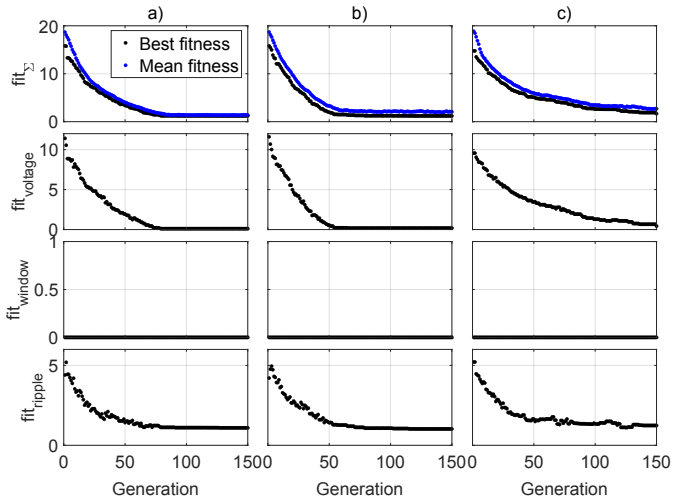

Fig. 4. Convergence of the GA with a) Binary- b) Grouped-binary- c) Integer-representation

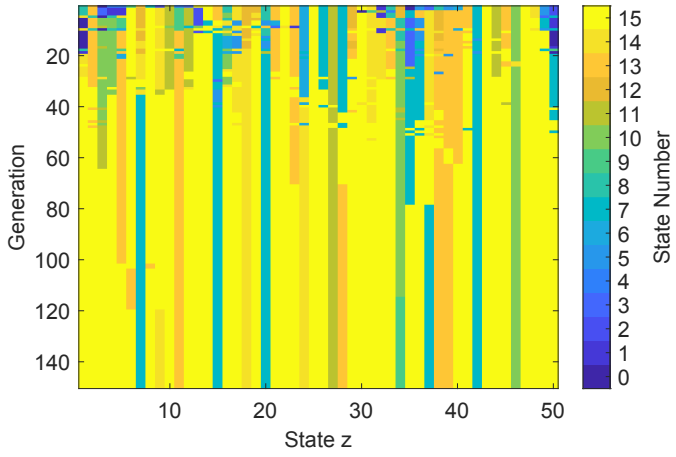

Fig. 5. Best chromosome in generation with grouped-binary-representation (color corresponds to state)

4.2 Optimized PWM pattern

The optimization result has to be translated back into a usable PWM pattern to be applied to each of the four switches. Fig. 6 shows the generated pattern within a time interval of 50 µs for a voltage of about 10 V per phase and a number of phase current measurement windows of about 5 windows. Fifty states were optimized to reach an average reference voltage, a given number of current measurement windows and minimal current ripple for each phase. The state number is shown for every state upper the time axis in the upper plot of Fig. 6.

4.3 Multi-objective optimization

In previous tests, a weighted fitness function was used. This requires a suitable choice of the weighting factors to achieve the qualitatively desired result. This often is a trial and error process which results in many test runs. The Global Optimization Toolbox from MathWorks provides a genetic algorithm named `gamultiobj` to solve multi-objective optimization problems by finding an evenly distributed set of points on the Pareto front. This approach also allows the optimization of nonsmooth prob-

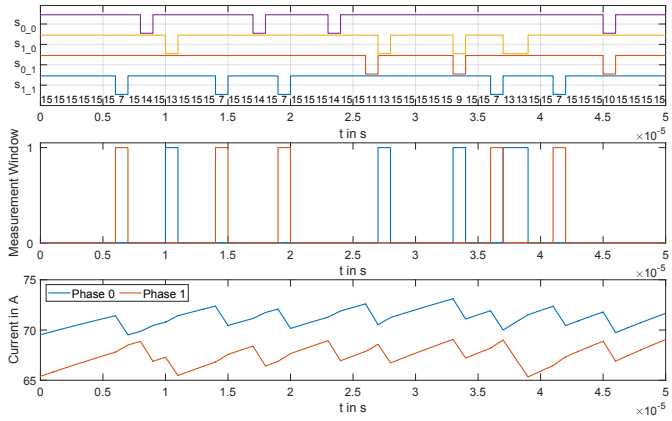

Fig. 6. PWM pattern after optimization

lems (MathWorks (2020a)). For the test, the grouped-binary-representation with the main part of the previous settings was chosen. The required phase voltages are 10 V for each phase and the number of phase current measurement windows should be 5. To achieve a good convergence, the number of generations had to be increased to 400. The selection function had to be changed to tournament selection, which is the only supported function in the `gamultiobj` algorithm. Two additional parameters - the Pareto fraction (0.5) and migration fraction (0.35) - were defined. Fig. 7 shows the Pareto front. It is characterized by three two-dimensional tradeoff curves for the discrete measurement window fitness values fit_{window} of 0, 1 and 2. Most individuals of the Pareto set lie on the inner curve where $fit_{window} = 0$. The red star marks the chosen optimal individual with the fitness values $fit_{voltage}$=0.179, fit_{window}=0 and fit_{ripple}=0.915. In comparison to the results of the previous tests where a weighted fitness function was used, all fitness values are equal to or smaller than those (for comparison grouped-binary-representation: $fit_{voltage}$=0.179, fit_{window}=0 and fit_{ripple}=1.026). This shows the efficiency of the multi-objective optimization and its application where a large number of requirements needs to be fulfilled (see list in section 2.2). The disadvantage of this approach is the increased computing time in comparison to a GA using a weighted fitness function. A possible combination of the two approaches is to use the found Pareto set to detect

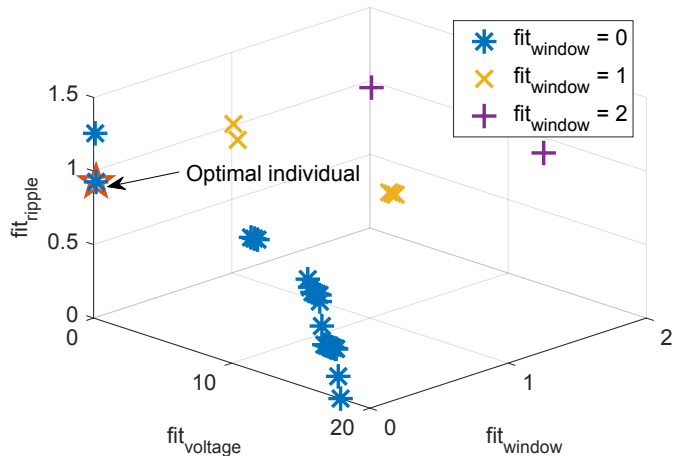

Fig. 7. Pareto front

suitable weighting factors for a classical GA. The design idea is to find a weighting factor set which leads to the smallest fitness value fit_Σ for the chosen optimal individual in comparison to all other individuals of the Pareto set.

4.4 Parallelization

To analyze the benefit of parallelization, the GA algorithm was run on a Windows 64-bit operating system using MATLAB 2016b on a Dell Precision T5600 Workstation with two Intel Xeon E5-2665 processors. This allows the usage of 16 processor cores. For the tests, the grouped-binary-representation with the Simulink system in Fig. 3 was used. All tests ran three times with a high priority set for the MATLAB process. Fig. 8 shows the computing time with different worker threads n_w. The ideal and the real speedup as well as modeled speedup curves $S(n_w)$ using Amdahl's law (adapted for GA)

$$S(n_w) = \frac{T_s + n_p T_e}{T_s + T_e \frac{n_p}{n_w}} \tag{6}$$

and a developed speedup equation for GA

$$S(n_w) = \frac{T_s + n_p T_e}{T_s + n_w T_c + T_{ep} \left\lceil \frac{n_p}{n_w} \right\rceil} \tag{7}$$

which combines the speedup equations of Cantú-Paz (2000) and Trobec (2009) are shown in Fig. 9. This equation considered all the observed effects by the parallel GA computation and allows a better modeling of the real speedup in comparison to Amdahl's Law. The usage of two individual values for the single fitness calculation time (serial execution: T_e, parallel execution: T_{ep}) allows the modeling of the superlinear speedup for a low number of workers ($n_w < 6$) as a result of a more efficient usage of resources using the Parallel Computing Toolbox ($T_e > T_{ep}$). These two values also acknowledge the fact that the fitness calculation time is dependent on the simulation time of the Simulink system which can vary. According to Trobec (2009), the value of T_{ep} describes the time for the concurrent execution of n_w fitness evaluations in a group and, hence, it corresponds to the longest period of the several fitness evaluations in the group. The serial start of all worker processes as well as the communication time between the GA (master) and the fitness evaluations (slaves) are considered by the product of $n_w T_c$ according to Cantú-Paz (2000). The ceiling function ($\lceil\rceil$) in the calculation of the split groups $\left\lceil \frac{n_p}{n_w} \right\rceil$ considers the fact that the last group with fewer fitness evaluation calls as available worker threads needs the same time as the other groups. The parameters in (6) and (7) are calculated from regression using the measured computing times. Table 3 includes the parameter descriptions and the calculated values. The analysis of (7) shows that with the increasing number of workers, the computation time for all fitness evaluations decreases while the communication time increases. The number of workers needed to approach the minimal execution time can be solved using $\frac{\partial S(n_w)}{\partial n_w} = 0$. For these tests, the optimal number n_w is 29, where a maximum speedup $S(n_w)$ of 14.75 will be achieved. An estimation of the necessary workers for an efficient work can be determined using the parallel efficiency

Table 3. Speedup equation parameters

Parameter	value
Population size n_p	200
Number of generations	150
Amdahl's Law (6)	
Single generation time without fitness calculations T_s	0.487 s
Single fitness evaluation time T_e	0.230 s
Equation for GA (7)	
Single generation time without fitness calculations T_s	0.132 s
Single fitness evaluation time (serial execution) T_e	0.236 s
Communication time T_c	0.053 s
Single fitness evaluation times (parallel execution) T_{ep}	0.222 s

$$E(n_w) = \frac{S(n_w)}{n_w} \tag{8}$$

Using an efficiency E of 0.8, which corresponds to an 80% performance of the parallel GA algorithm of its ideal potential, results in a number of workers n_w = 15.6 or rather 15. This means the usage of more than 15 workers does not lead to any significant computing speedup effects. The equations presented above and the relationships allow a rough estimation of the necessary hardware to solve GA algorithms efficiency in terms of time and cost.

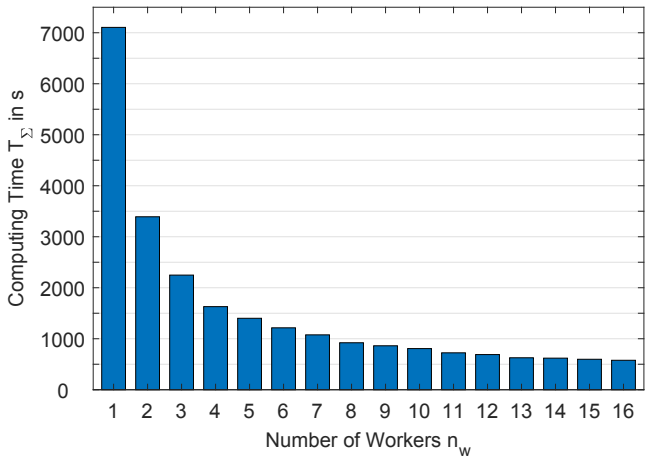

Fig. 8. Computing time using several worker threads

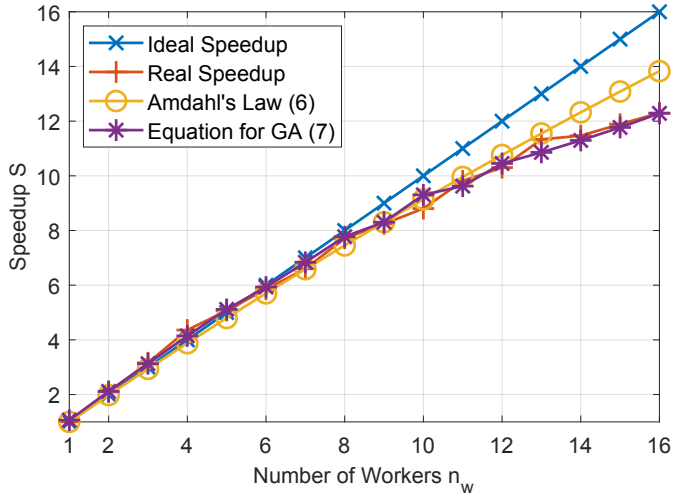

Fig. 9. Ideal, real and modeled speedup curves

5. CONCLUSION

This paper presented a concept to generate control sequences for a SRM with a reduced amount of current sensors using GA. Different genotype representations were tested for increasing the convergence rate. To fulfill all requirements, a weighted multi-objective fitness function as well as a Pareto front solution were used successfully. An approach to determine the number of cores to solve GA algorithms efficiently was presented. The software implementation based on MATLAB is discussed in detail.

REFERENCES

Babatunde, O. (2020). Binary_genetic_algorithm_hezy_2013. URL `https://www.mathworks.com/matlabcentral/fileexchange/46961-binary_genetic_algorithm_hezy_2013`.

Babatunde, O., Armstrong, L., Leng, J., and Dean, D. (2014). A genetic algorithm-based feature selection. *International Journal of Electronics Communication and Computer Engineering*, 5(4), 889–905.

Cantú-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*, volume 1. Kluwer Academic Publ, Boston, Mass.

Chen, H. and Lu, S. (2013). Fault diagnosis digital method for power transistors in power converters of switched reluctance motors. *IEEE Transactions on Industrial Electronics*, 60(2), 749–763. doi:10.1109/TIE.2012.2207661.

Kjaer, P.C. and Gallegos-Lopez, G. (1997). Single-sensor current regulation in switched reluctance motor drives. In *IAS '97. Conference Record of the 1997 IEEE Industry Applications Conference Thirty-Second IAS Annual Meeting*, volume 1, 645–652 vol.1. doi:10.1109/IAS.1997.643138.

MathWorks (2020a). Global Optimization Toolbox. URL `https://www.mathworks.com/products/global-optimization/`.

MathWorks (2020b). Parallel Computing Toolbox. URL `https://www.mathworks.com/products/parallel-computing/`.

Plexim (2020). Plecs Blockset - The Power Electronics Toolbox for Simulink. URL `https://www.plexim.com/products/plecs_blockset`.

Prabhu, V.V., Rajini, V., Balaji, M., and Prabhu, V. (2015). A comparative study of operating angle optimization of switched reluctance motor with robust speed controller using pso and ga. *Journal of Electrical Engineering and Technology*, 10(2), 551–559. doi:10.5370/JEET.2015.10.2.551.

Purfürst, S. (2019). *Beitrag zur schaltungszustandsbasierten Ansteuerung geschalteter Reluktanzmaschinen unter Berücksichtigung der Strommessung bei reduzierter Sensorik*. Ph.D. thesis, TU Ilmenau.

Shaked, N.T. and Rabinovici, R. (2006). Structural switched reluctance motor optimization using the genetic algorithm. *Journal of Electrical Engineering*, 6(4).

Sihem, M., Amar, B., and Hocine, B. (2012). Teeth shape design of a switched reluctance motor for high torque using genetic algorithms. *International Journal of Scientific & Engineering Research*, 3(10), 1128–1132.

Trobec, R. (2009). *Parallel Computing - Numerics, Applications, and Trends*. Springer, Dordrecht.