Comparison of Parallel Metaheuristics for flux optimization for Induction Motor

VINCENT ROBERGE, MOHAMMED TARBOUCHI
Electrical and Computer Engineering
Royal Military College of Canada
PO Box 17000, Station Forces, Kingston, Ontario, K7K 7B4
CANADA
vincent.roberge@rmc.ca, tarbouchi-m@rmc.ca

Abstract: - An essential aspect of efficiency control of a three-phase induction motor is the ability to generate the optimal magnetic flux required for different operating modes. In this paper, we use the genetic algorithm (GA), the particle swarm optimization algorithm (PSO) and the simulated annealing (SA) to cope with the complexity of the problem and compute feasible and quasi-optimal magnetic flux needed for three-phase induction motors with time varying load and parameters. The characteristics of the optimal magnetic flux are represented in the form of a multi-objective cost function that we developed. We reduce the execution time of our solutions by using the "single-program, multiple-data" parallel programming paradigm and achieve real-time performance on a multi-core CPU.

Key-Words: -magnetic flux optimization, genetic algorithm, particle swarm optimization, simulated annealing, parallel computation

1 Introduction

For the control of an electric induction motor (IM) variable speed, the magnetic flux must be adjusted to obtain the best possible efficiency. The optimal flux depending on the desired speed and torque is often calculated using deterministic methods that require considerable computing power. For this reason, the calculation of magnetic flux is usually done offline. This approach may not guarantee the optimal expected performances because the state of the machine changes during operation (temperature, saturation and other constraints). As presented in [1], [2] and [3], there are ways to reassess the characteristics or parameters of the IM. It would therefore be advantageous to calculate the optimal flux in real time to continually enhance performance even when the motor parameters change. To minimize the computing time and to allow embedded application, non-deterministic algorithms seems excellent candidates for optimal flux generation [4], [5], [6] and [7].

In this work, we develop and evaluate three non-deterministic optimization algorithms to calculate the flux maximizing the effectiveness of the induction motor. To reduce the computation time and allow real time performance, we also implement and compare parallel versions of the three algorithms. The remainder of this paper is organized as follows. Section 2 deals with magnetic flux control of an induction motor and identifies the objective function to be optimized. Section 3 presents the three optimization algorithms used: the genetic algorithm, the particle swarm optimization and simulated annealing. Section 4 discusses our parallel implementation of the three algorithms. Finally, a comparison of results obtained by each algorithm is presented in Section 5.

2 Optimizing the magnetic flux of an induction motor

Today, the induction motors are the most used machines in variable speed drives. For this reason, several researchers linger to develop control methods that optimize the efficiency of these machines. As explained in [8], one of the challenges in controlling induction motor whether using scalar, vector control or other modern nonlinear techniques is the generation of the optimal magnetic flux required for different operating conditions. The control modules commonly used normally maintain the magnetic flux close to nominal values thus ensuring good efficiency when the motor operates at a speed and torque near the nominal point. However, the efficiency decreases greatly when the speed or torque varies. To improve efficiency, it is important to adjust the flux when the speed or

torque changes. Still according to [8], two approaches are mainly used: the method based on the loss model and that based on the measurement of power. In this paper we use the first approach and show that non-deterministic optimization algorithms (GA, PSO and SA) can be used to calculate the optimal flux which minimizes the loss of the IM for a given speed and load torque.

2.1 Loss model of induction motor

The synchronous reference frame model of a three-phase IM is used [8]. The block diagram of the control system under study is shown in Fig. 1. For an angular velocity and torque points, the control module uses an optimization algorithm that calculates the optimum flux i_u to minimize the IM losses equation.

The losses in the IM are mainly composed of rotor copper losses (P_{jr}), copper losses in the stator (P_{js}), iron losses (P_{fe}) and mechanical losses (P_m). The losses of IM are calculated as follows:

$$P_{loss} = P_{js} + P_{jr} + P_{fe} + P_m \tag{1}$$

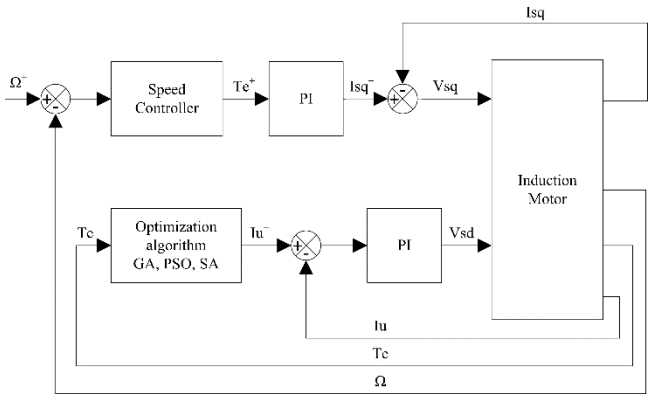

Fig. 1. Block diagram of the three-phase IM control system

Since the mechanical losses are not related to magnetic flux, they are omitted in calculation the optimal flux. This same equation is presented in [9] as follows where each term is defined in Table 1:

$$P_{loss}(i_d, i_q, \omega_e) = \left(R_s + \frac{\omega_e^2 L_m^2}{R_m}\right) * i_d^2 + \left(R_s + \frac{R_r L_m^2}{L_r^2} + \frac{\omega_e^2 L_m^2 L_{lr}^2}{R_m L_r^2}\right) * i_q^2 \tag{2}$$

It is important to note that the synchronous angular velocity ω_e can be approximated by the angular velocity of the motor ω_r when the slip is negligible (light load conditions). Still according to [9], the magnetic flux i_u and electromagnetic torque T_e are defined as follows:

$$i_u = \sqrt{i_d^2 + i_q^2} \tag{3}$$

$$T_e = \frac{3}{2} p \frac{L_m^2}{L_r} * i_d i_q \tag{4}$$

The optimization problem of the magnetic flux as a function of torque and speed is to find values for i_d and i_q that minimize equation (2) while generating an electromagnetic torque T_e greater or equal to the required torque T_r. In addition, i_d and i_q must meet two requirements: that of the maximum current and maximum voltage.

$$i_d^2 + i_q^2 \leq I_{max}^2 \tag{5}$$

$$(\omega_e L_s i_d)^2 + (\omega_e \sigma L_s i_q)^2 \leq V_{max}^2 \tag{6}$$

where $$\sigma = 1 - \frac{{L_m}^2}{L_s L_r} \tag{7}$$

I_{max} is the maximum current possible and V_{max} is the maximum voltage possible. It is then possible to represent the loss of IM, the electromagnetic torque and the two constraints as a cost function that we minimize using optimization algorithms described the next section. In this cost function, i_d and i_q are the variables to be optimized for given angular speed ω_e and electromagnetic torque T_{req} given:

$$F_{cost}(i_d, i_q) = P_{loss}(i_d, i_q, \omega_e) + Penalty_{Treq} + Penalty_{Imax} + Penalty_{Vmax} \tag{8}$$

where

$$Penalty_{T_{req}} = \begin{cases} \frac{T_{req}}{T_e} * 10^9, & if \quad T_e < T_{req} \\ 0, & if \quad T_e \geq T_{req} \end{cases} \tag{9}$$

$$Penalty_{I_{max}} = \begin{cases} 0, & if \quad i_d^2 + i_q^2 \leq I_{max}^2 \\ \frac{i_d^2 + i_q^2}{I_{max}^2} * 10^9, & if \quad i_d^2 + i_q^2 > I_{max}^2 \end{cases} \tag{10}$$

$$Penalty_{V_{max}} = \begin{cases} 0, & if \quad V^2 \leq V_{max}^2 \\ \frac{(\omega_r L_s i_d)^2 + (\omega_r \sigma L_s i_q)^2}{V_{max}^2} * 10^9, & if \quad V^2 > V_{max}^2 \end{cases} \tag{11}$$

where

$$V^2 = (\omega_e L_s i_d)^2 + (\omega_e \sigma L_s i_q)^2 \tag{12}$$

We use a very large constant (here $1 * 10^9$) in equations(9), (10) and (11) to separate the valid

solutions (those that produce the necessary torque and respect the constraints) from the invalid ones. This penalty is added in proportion to the degree ofviolation of constraints and thus allows the improvement of invalid solution in order to generate valid ones. In this paper, all calculations are done using the parameters used in [8] and [10] whose values are reproduced in Table 1.

Table 1. Parameters of the IM used in this document

Parameter	Symbols	Values	Units
Power	P	1100	W
Speed	ω_n	157.08	rad/sec
Voltage	V_{max}	220 à 380	V
Current	I_{max}	3.4	A
Rated load torque	T_n	7	N*m
Number of pairs of poles	p	2	
Stator resistance	R_s	8	Ω
Rotor resistance	R_r	3.1	Ω
Total leakage factor $(\sigma = 1 - L_m^2/(L_s L_r))$	σ	0.12	N/A
Mutual inductance	L_m	0.443	H
Stator leakage inductance	L_{ls}	0.027	H
Rotor leakage inductance	L_{lr}	0.027	H
Stator self-inductance $(L_s = L_{ls} + L_m)$	L_s	0.47	H
Rotor self-inductance $(L_r = L_{lr} + L_m)$	L_r	0.47	H
Inertia	J	0.06	SI
Viscous friction coefficient	f	0.00	SI

3. Optimization algorithms

In this section we present three optimization algorithms that we implemented to find the values i_d and i_q that minimize the cost function defined in equation (8).

3.1 Genetic algorithm

The GA is a non-deterministic optimization method based on a population and was developed by John Holland in the '60s and first published in 1975 [11]. Based on the theory of Darwinian evolution, the GA simulates the evolution of a population of solutions to optimize a problem.Like living organisms adapt to their environment, the solutions of the GA to adapt to the cost function in an iterative process that simulates the crossover and mutation of genes.In our implementation, we use a binary encoding and randomly generate a population of possible solutions i_d and i_q. The GA then modifies this population of candidate solution for a near-optimal final solution. The selection mechanism used is the stochastic universal sampling based on the ranking of solutions [12]. We also use the principle of elitism in order to improve the convergence of the algorithm [12]. The flow chart of the GA and the genetic operators used are shown in Fig. 2 and Fig. 3.

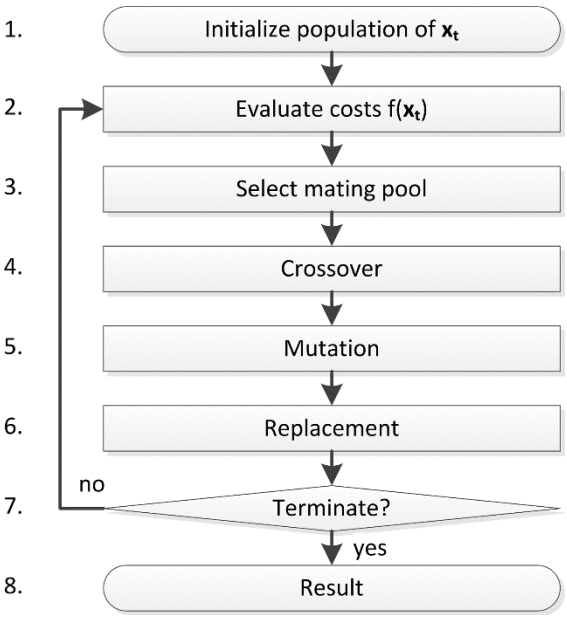

Fig. 2. Flow chart for the genetic algorithm

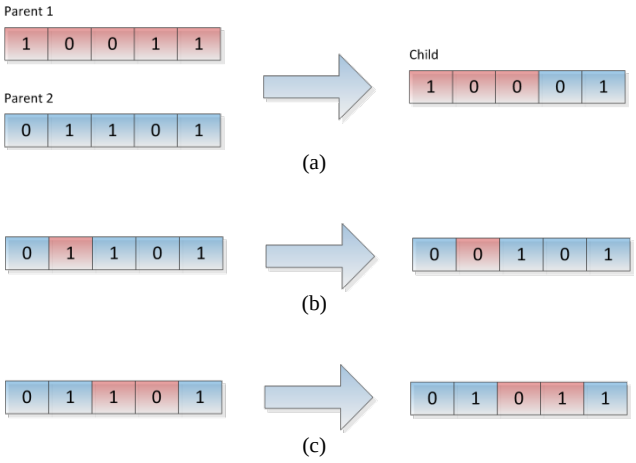

Fig.3. Genetic operators used: (a) single-point crossover, (b) bit-flip mutation, (b) bit-swap mutation

3.2 Particle swarm optimization

The PSO is a non-deterministic optimization method also based on population and was developed by Kennedy and Eberhard in 1995 [13]. The algorithm simulates the motion of a swarm of particles in a search space of one or more dimensions to an optimal position. The position of each particle represents a candidate solution and is initialized randomly.At each step of the iterative procedure, the

particle velocity is calculated individually based on the previous speed (inertia), the best position ever occupied by the particle (personal influence) and the best position ever occupied by any particle of swarm (social influence).

As defined in [14], the equations to update the velocity and position of a particle at iteration t are as follows:

$$\boldsymbol{v}_{t+1} = \omega \boldsymbol{v}_t + c_1 \boldsymbol{r}_1 .* (\boldsymbol{b}_t - \boldsymbol{x}_t) + c_2 \boldsymbol{r}_2 .* (\boldsymbol{g}_t - \boldsymbol{x}_t) \quad (13)$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_{t+1} \quad (14)$$

where variables in bold are vectors; $\boldsymbol{v}$ is the velocity of a particle; $\boldsymbol{x}$ is its position; $\boldsymbol{b}$ is the best position ever occupied by the particle; $\boldsymbol{g}$ is the best position ever occupied by any particle of the swarm; $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are vectors of random values between 0 and 1; and ω, c_1 and c_2 are the parameters of inertia, personal influence and social influence. Still based on [14], the operating diagram of the PSO is illustrated in Fig. 4.In our implementation, the particles move in a space of two dimensions and their position represents candidate values for i_d and i_q.

3.3 Simulated annealing

Presented for the first time in 1983 [15], simulated annealing is a non-deterministic optimization method which simulates the annealing process in which atoms in a heated metal escape their local minimum to eventually settle in an energy level lower than initially. Similarly to the PSO, one possible solution is encoded as an atom whose state varies in a space of one or more dimensions. By cons, optimization by simulated annealing uses only one particle.

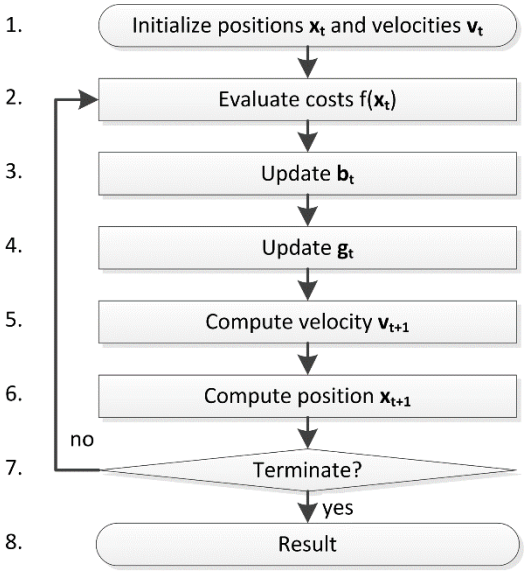

Fig.4. Flow chart for the particle swarm optimization

At the beginning of the process, when the temperature is high, the particle changes its state almost randomly allowing a higher cost solution to encourage exploration. As the process progresses and the temperature drops, this randomness decreases and the particle is directed primarily towards a solution that minimizes the cost function. As defined in [16], the probability of accepting a better solution is always 1, but defined as follows for an inferior solution:

$$P_{acceptance} = e^{-\frac{F(x(t+1)) - F(x(t))}{T}} \quad (15)$$

where $F(\boldsymbol{x}(t))$is the cost of the current solution $\boldsymbol{x}(t)$,$F(\boldsymbol{x}(t+1))$is the cost of the candidate solution $\boldsymbol{x}(t+1)$ and T is the temperature and is reduced every iteration following:

$$T = T_0 * \propto^{iteration} \quad (16)$$

where T_0 is the initial temperature and $\propto$ is a constant between 0 and 1. In our implementation, the state of the atom has two dimensions and represents a value ofi_d and i_q. We calculate the value of T_0 following the method presented in [17] so that a worse solution is initially accepted with a probability of about 95%. We also calculate the value $\propto$ so that the final temperature is equal to $0.01 * T_0$. Still according to[16], the flow char for the simulated annealing is shown in Fig. 5.

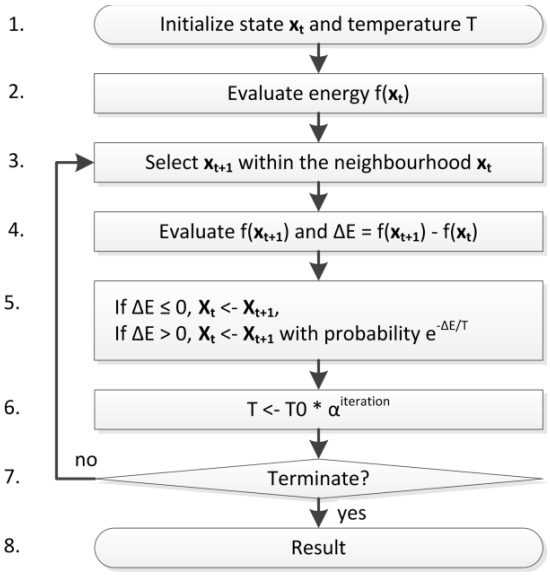

Fig.5. Flow chart for the simulated annealing

4. Parallel Implementation

In this paper, we selected three non-deterministic algorithms in order to cope with the complexity of

flux optimization for IM and produce quasi-optimal solutions in a very short computation time. Since multicore CPUs have become moreand more common in personal computersand microcontrollers,the execution time can be reduced whenusing parallel implementations. In this section, we present our approach to parallelizing the GA, the PSO and the SA using the "single-program, multiple-data" parallel programming paradigm. Our implementation is done using the MATLAB™ Parallel Computing Toolbox 6.0

4.1 Parallel GA

Different approaches have been proposed to parallelized the GA. They can be classified as: master-slave, coarse-grain, fine-grain and hybrid [18]. As discussed in [19], a coarse-grain approach simulating the evolution of independent populations is usually preferable for a multi-core execution. In our parallel implementation of the GA, we divide the population by the number of processes and each process simulates an independent swarmbased on the flow chart in Fig. 2. Our implementation allows communication between the populations by a process of migration where the best solutions from process *i* are transmitted to process *i + 1* where they will replace the worse solutions. This migration takes place between steps 6 and 7 in Fig. 2 and occurs every fifth iteration. As explained in [20], allowing few generations between migrations slows down convergence, improves exploration and reduces the inter-process communication. Our implementation results in a very good speedup due to the parallelization of all stages of the algorithm and the minimization of communication.

4.2 Parallel PSO

Our parallel implementation of the PSO follows the parallel broadcast method discussed in [21] and is very similar to our parallel implementation of the GA. We divide the swarm between the processes and each process simulates the movement of an independent swarm. Every fifth iteration, we compare the best particle of each swarm in order find the global best particle. This global best particle is then broadcasted to all swarms. As for the GA, this implementation parallelizes every step of the algorithm and minimizes communication resulting in a superior speedup.

4.3 Parallel SA

Unlike the GA and the PSO, the SA is not a population based algorithm and is limited to the simulation of a single solution. In order to maintain the essence of the algorithm, our parallel implementation of the SA also uses a single atom or solution per process with no communication between the processes. At the end of program, the costs of the solutions produced by the different processes are compared the best solution is returned. Due to the sequential nature of the SA algorithm, our parallel implementation does not reduce the computation speed, but improves the quality of the final solution.

5.0 Results

We present in this section, the results obtained by our sequential and parallel version of the three algorithms discussed above. We compare those results to a brute force search that sequentially checks all possible values of i_d and i_q within the search space. When testing i_d and $i_q$from 0.5 A to 3.5 A using an increment of 0.01 A, the brute force algorithm evaluates 90 000 candidate solutions before returning the best one. To verify the accuracy of the GA, the PSO, the SA and the brute force search, we compare the results to a reference. This reference is generated using a brute force approach with a step of 0.0001 A resulting in an execution time of more than 2 hours. The configuration parameters used for each algorithm are shown in Table 2. All test are performed on a computer equipped with aquad-core Intel Xeon E3 1230 at 3.2 GHz and 4 GB DDR3 1333 MHz RAM.

Table 2. Configuration parameters for the different optimization algorithms implemented

Algorithms	Configuration parameters	Values
GA	Chromosome bit width	10
	Population size	32
	Number of generation	100
	Mutation rate	5 %
	Elitism rate	5 %
	Generations between migrations	5
	Number of chromosomes migrating	1
PSO	Swarm size	32
	Number of iterations	100
	ω	0.7298
	c_1	1.4960
	c_2	1.4960
	Iterations between global best particle broadcast	5
SA	Initial probability of accepting an inferior solution	95%
	Final temperature	0.01 * $T_{initial}$
	Number of iteration	2000
Brute-force	Increment when searching for	0.01 A

search	optimal i_d and i_q	

Before we present our results, it is interesting to visualize an example of the losses for different magnetic flux. Fig. 6 shows the operating losses versus i_d and i_q for an angular speed of 150 rad/s and a required torque of 5 N*m. It is important to note that the surface is limited by the constraints presented at equations (9), (10) and (11) and only shows the losses associated with values of i_d and i_q that generate the required electromagnetic torque without exceeding the maximum current and voltage of the IM.

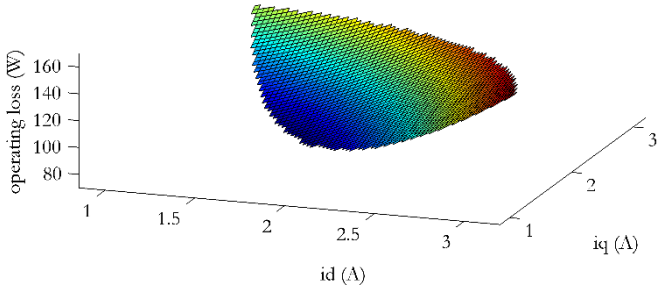

Fig.6. Losses versus i_d and i_q for ω_r = 150 rad/s and T_e = 5 N*m

We then use the GA, the PSO, the SA and the brute force search to find the optimal values i_d and i_q that minimize the equation (8). The calculated values of i_d,$i_q$and associated losses for different speeds and loads are listed in Table 3(average of 1000 trials). We also present in this table the differences between the results obtained by our optimization algorithms and the reference. The best results are printed in bolds and show that the PSO generally produced solutions that are closer to the reference and that yield smaller losses than the other three algorithms. We also note that the PSO took 0.05 s to execute which is 20x faster than for the GA, 1.8x faster than the SA and 17xfaster than for the brute force search.

Next, we repeat this test using our parallel versions of the GA, the PSO and the SA. In order to provide a fair comparison of the execution time, we also parallelized the brute force search. Our implementation divides the search space by the number of processes and each process performs a search on its subspace. Once completed, the results produced by each process are compared and the best solution is returned. Although not limited to 4 processes, all algorithms are run using 4 processes in order to maximize the efficiency on ourquad-core Intel Xeon E3 1230 CPU. The calculated values of i_d,$i_q$and associated losses for different speeds and loads are listed in Table 4(average of 1000 trials). The best results are printed in bolds andstill show that the PSO generally produced solutions that are closer to the reference and that yield smaller losses than the other three algorithms. We also measure a speedup of 3.35x for the PGA, 2.51x for the PPSO and 3.57x for the brute force search. As discussed earlier, the PSA is not expected to provide any speedup. The PPSO remains the preferable approach producing the best solution in the shortest execution time. In a real time application, the PPSO would have the ability to generate the optimal magnetic flux required for different operating modes at a frequency of 52.6 Hz which would not be possible with any the other approaches discussed here. This makes the PPSO the best solution for real-time flux optimization on multi-core CPU.

Finally, we use the PPSO to calculate the optimal values i_d and i_q(Fig. 7 and Fig. 8) and associated losses (Fig. 9) for speeds ranging from 0 to 300 rad/s and varying loads from 1 to 7 N*m (157 rad/s and 7 N*m are the nominal values). As we saw in Table 4, the PGA, the PSA and the parallel brute force search produce results very close to the PPSO and the generated surfaces are almost identical to those of Fig. 7 and Fig. 9.

Table 3. Flux with associated losses as computed by the Sequential GA, the Sequential PSO, the Sequential SA and the Sequential brute force search for different speeds and torques (Average of 1000 trials)

Variables	Torque, speed (N*m, rad/s)	Optimization Algorithms				
		GA	PSO	SA	Brute-force search	Reference
I_d (A)	**4, 100**	1.645	1.660	1.661	1.690	1.6598
	4, 150	1.498	1.486	1.488	1.500	1.4863
	6, 100	2.031	2.033	2.036	2.030	2.0328
	6, 150	1.902	1.821	1.824	1.850	1.8203
I_q (A)	**4, 100**	1.961	1.924	1.927	1.890	1.9238
	4, 150	2.147	2.148	2.151	2.130	2.1485
	6, 100	2.366	2.356	2.357	2.360	2.3562
	6, 150	2.538	2.630	2.631	2.590	2.6313
Losses (W)	**4, 100**	80.966	79.767	79.996	79.836	79.7635
	4, 150	100.689	99.728	100.007	99.794	99.7228
	6, 100	120.192	119.651	119.903	119.670	119.6452
	6, 150	151.799	149.591	149.902	149.713	149.5841
$\lvert I_d^{ref} - I_d \rvert$ (A)	**4, 100**	0.015	**0.000**	0.002	0.030	N/A
	4, 150	0.011	**0.000**	0.002	0.014	N/A
	6, 100	0.002	**0.000**	0.003	0.003	N/A
	6, 150	0.082	**0.001**	0.003	0.030	N/A
$\lvert I_q^{ref} - I_q \rvert$ (A)	**4, 100**	0.037	**0.000**	0.003	0.034	N/A
	4, 150	0.002	**0.000**	0.003	0.018	N/A
	6, 100	0.010	**0.000**	**0.000**	0.004	N/A
	6, 150	0.094	0.001	**0.000**	0.041	N/A
Execution time (s)	**4, 100**	1.008	**0.049**	0.087	0.840	8 762.4
	4, 150	1.001	**0.048**	0.086	0.832	8 643.5
	6, 100	1.001	**0.049**	0.086	0.834	8 635.2
	6, 150	1.003	**0.049**	0.086	0.835	8 659.7

Table 4. Flux with associated losses as computed by the Parallel GA, the Parallel PSO, the Parallel SA and the Parallel brute force search for different speeds and torques (Average of 1000 trials)

Variables	Torque, speed (N*m, rad/s)	Optimization Algorithms				
		PGA	PPSO	PSA	P Brute-force search	Reference
I_d (A)	**4, 100**	1.644	1.660	1.660	1.690	1.6598
	4, 150	1.501	1.486	1.486	1.500	1.4863
	6, 100	2.020	2.032	2.034	2.030	2.0328
	6, 150	1.859	1.822	1.821	1.850	1.8203
I_q (A)	**4, 100**	1.956	1.923	1.925	1.890	1.9238
	4, 150	2.137	2.149	2.151	2.130	2.1485
	6, 100	2.375	2.357	2.356	2.360	2.3562
	6, 150	2.587	2.630	2.632	2.590	2.6313
Losses (W)	**4, 100**	80.598	79.767	79.854	79.836	79.7635
	4, 150	100.445	99.728	99.831	99.794	99.7228
	6, 100	119.975	119.650	119.747	119.670	119.6452
	6, 150	150.890	149.591	149.706	149.713	149.5841
$\lvert I_d^{ref} - I_d \rvert$ (A)	**4, 100**	0.016	**0.000**	0.000	0.030	N/A
	4, 150	0.014	**0.000**	0.000	0.014	N/A
	6, 100	0.013	**0.000**	0.001	0.003	N/A
	6, 150	0.039	**0.001**	**0.001**	0.030	N/A
$\lvert I_q^{ref} - I_q \rvert$ (A)	**4, 100**	0.032	**0.000**	0.001	0.034	N/A
	4, 150	0.011	**0.000**	0.002	0.018	N/A
	6, 100	0.018	**0.001**	**0.001**	0.004	N/A
	6, 150	0.044	0.002	**0.001**	0.041	N/A
Execution time (s)	**4, 100**	0.288	**0.020**	0.088	0.229	8 762.4
	4, 150	0.293	**0.019**	0.090	0.231	8 643.5
	6, 100	0.307	**0.019**	0.088	0.260	8 635.2
	6, 150	0.307	**0.019**	0.088	0.214	8 659.7

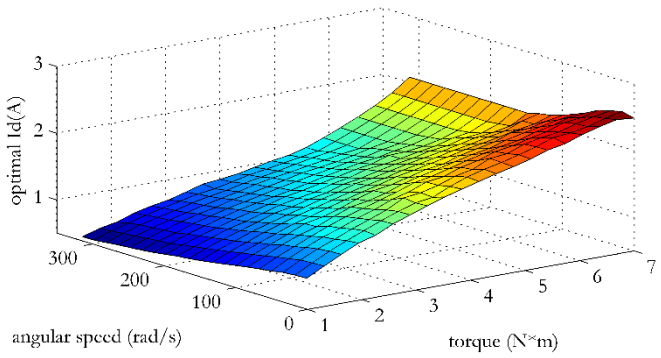

Fig.7. Optimal i_d for different speed and torque as computed by the PPSO

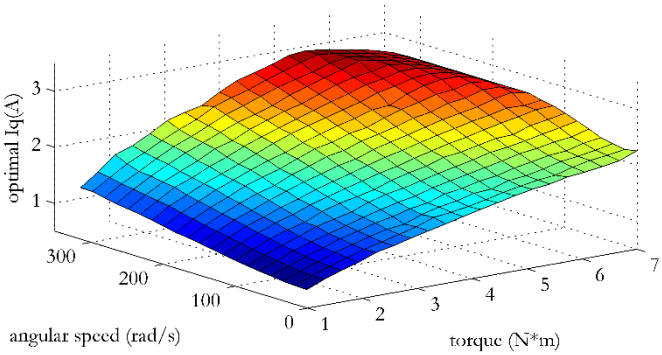

Fig.8. Optimal i_q for different speed and torque as computed by the PPSO

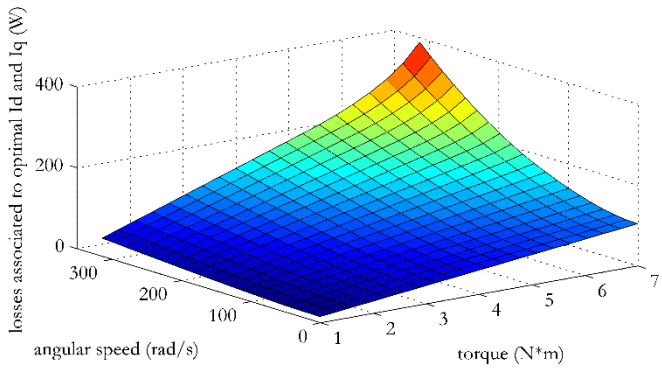

Fig.9. Losses associated with the optimal i_d and i_q for different speed and torque as computed by the PPSO

9. Conclusion

In this paper we have shown how the GA, the PSO and the SA can be used to calculate the value of the magnetic flux in induction motor for optimal efficiency. We have also shown how the execution time of the GA and PSO can be significantly reduced using a parallel implementation on a multi-core CPU. The PPSO is the most efficient algorithm that calculates the reference magnetic flux in 0.02 s with accuracy better than ±0.002A. With the obtained computation time, it is no longer necessary to use off line look-up tables for magnetic flux generation, even when the induction motor parameters changes.

References:

[1] M. Mutluer, O. Bilgin, and M. Cunkas, "Parameter determination of induction machines by hybrid genetic algorithms," 11th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, KES 2007, and 17th Italian Workshop on Neural Networks, WIRN 2007,

September 12, 2007 - September 14, 2007, Vietri sul Mare, Italy: Springer Verlag, 2007, pp. 116–124.

[2] H.M. Emara, W. Elshamy, and A. Bahgat, "Parameter identification of induction motor using modified particle swarm optimization algorithm," 2008 IEEE International Symposium on Industrial Electronics (ISIE 2008), 30 June-2 July 2008, Piscataway, NJ, USA: IEEE, 2008, p. 841–7.

[3] C. Guangyi, G. Wei, and H. Kaisheng, "On line parameter identification of an induction motor using improved particle swarm optimization," 26th Chinese Control Conference, CCC 2007, July 26, 2007 - July 31, 2007, Zhangjiajie, China: Inst. of Elec. and Elec. Eng. Computer Society, 2007, pp. 745–749.

[4] D.H. Kim, "GA–PSO based vector control of indirect three phase induction motor," Applied Soft Computing, vol. 7, Mar. 2007, pp. 601–611.

[5] E. Poirer, M. Ghribi, and A. Kaddouri, "Loss minimization control of induction motor drives based on genetic algorithm," Cambridge, MA: 2001, pp. 475–478.

[6] L.R. Valdenebro and E. Bim, "A genetic algorithms approach for adaptive field oriented control of induction motor drives," Proceedings of Electric Machines and Drives Conference, 9-12 May 1999, Piscataway, NJ, USA: IEEE, 1999, p. 643–5.

[7] E.S. Abdin, G.A. Ghoneem, H.M.M. Diab, and S.A. Deraz, "Efficiency optimization of a vector controlled induction motor drive using an artificial neural network," Industrial Electronics Society, 2003. IECON '03. The 29th Annual Conference of the IEEE, 2003, pp. 2543– 2548 Vol.3.

[8] Z. Rouabah, F. Zidani, and B. Abdelhadi, "Efficiency optimization of induction motor drive using fuzzy logic and genetic algorithms," 2008 IEEE International Symposium on Industrial Electronics (ISIE 2008), 30 June-2 July 2008, Piscataway, NJ, USA: IEEE, 2008, p. 737–42.

[9] S. Lim and K. Nam, "Loss-minimising control scheme for induction motors," IEE Proceedings: Electric Power Applications, vol. 151, 2004, pp. 385–397.

[10] E. Mendes, A. Baba, and A. Razek, "Losses minimization of a field oriented controlled induction machine," Seventh International Conference on Electrical Machines and Drives, 11-13 Sept. 1995, London, UK: IEE, 1995, p. 310–14.

[11] J.H. Holland, Adaptation in Natural and Artificial Systems, 1975.

[12] X. Yu and M. Gen, Introduction to Evolutionary Algorithms, London: Springer, 2010.

[13] J. Kennedy and R. Eberhart, "Particle swarm optimization," Proceedings of the 1995 IEEE International Conference on Neural Networks. Part 1 (of 6), November 27, 1995 - December 1, 1995, Perth, Aust: IEEE, 1995, pp. 1942–1948.

[14] M. Clerc, Particle Swarm Optimization, London: ISTE, 2006.

[15] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," May. 1983, pp. 671–680.

[16] E.-G. Talbi, Metaheuristics, From Design to Implementation, New Jersey: Wiley, 2009.

[17] C.M. Tan, Simulated Annealing, Vienna, Austria: I-Tech, 2008.

[18] S.N. Sivanandam and S.N. Deepa, Introduction to Genetic Algorithms, Berlin Heidelberg New York: Springer, 2008.

[19] Long Zheng, Yanchao Lu, Mengwei Ding, Yao Shen, Minyi Guoz, and Song Guo, "Architecture-based Performance Evaluation of Genetic Algorithms on Multi/Many-core Systems," Computational Science and Engineering (CSE), 2011 IEEE 14th International Conference on, 2011, pp. 321–334.

[20] B.T. Skinner, H.T. Nguyen, and D.K. Liu, "Performance Study of a Multi-Deme Parallel Genetic Algorithm with Adaptive Mutation," Palmerston North, New Zealand: 2004.

[21] K.-Y. Tu and Z.-C. Liang, "Parallel computation models of particle swarm optimization implemented by multiple threads," Expert Systems with Applications, vol. 38, May. 2011, pp. 5858–5866.