Evolutionary Algorithms for Induction Motor Parameter Determination

Phaderm Nangsue, Pragasen Pillay and Susan E. Conry
ECE Department, Clarkson University, Potsdam NY. 13699-5720
nangsue@sun.soe.clarkson.edu, pillayp@sun.soe.clarkson.edu, conry@clarkson.edu
(315) 268-6511

Abstract-This paper demonstrates the applicability of evolutionary algorithms to the problem of motor parameter determination. Motor parameter determination problems can range from high accuracy requirement for motor controlled drives to low accuracy requirements for system studies. The later problem is addressed here, using both the genetic algorithm and genetic programming. Comparative results are presented.

1. Introduction

There are many occasions where knowledge of motor parameters for system type studies are needed, albeit not at the level of accuracy needed for accurate efficiency determination for example. The source data for determination of such parameters is also often generic, such as torque data from a manufacturer describing a whole range of machines of a certain design. The equations relating the required motor parameters to the given data are often nonlinear in nature. Optimization techniques like the Newton-Raphson techniques have been applied to this type of problem with some success, although with the inherent problem of convergence to a local minimum instead of the global minimum. The optimum determined by the Newton-Raphson technique depends heavily on the initial guess of the parameter, with the possibility of a slightly different initial value causing the algorithm to converge to an entirely different solution. One important parameter required by the algorithm is the derivative of the function, which is not always available or may be difficult to calculate.

These problems have encouraged the authors to investigate alternative techniques of solution. One of these is evolutionary algorithms. These methods encompass a broad class of algorithms, two of which are the genetic algorithm and the genetic programming techniques [1-12].

The paper is organized as follows: Section 2 presents the fundamental principles of genetic methods. Section 3 applies them to the problem of motor parameter determination. Section 4 contains our experimental results and concluding remarks are found in section 5.

PE-256-EC-0-2-1998 A paper recommended and approved by the IEEE Electric Machinery Committee of the IEEE Power Engineering Society for publication in the IEEE Transactions on Energy Conversion. Manuscript submitted July 31, 1997; made available for printing March 2, 1998.

2. Fundamental of Genetic Methods

2.1 Genetic Algorithms

Genetic algorithms (GAs) are weak methods of problem solving inspired by natural genetics. GAs manipulate strings of binary digits and measure each string's strength using fitness values. The stronger strings are retained and recombined with other strong strings to produce offspring. Weaker ones are discarded. Eventually one string emerges as the best. There are many differences between GAs and conventional optimization techniques as shown in Table 2.1.

Compared to conventional techniques, the GAs' advantages are

1) GAs are domain independent and therefore can be applied to various problems.

2) GAs require none of the problem-specific auxiliary knowledge such as derivatives and good initial guesses. This can be beneficial especially when dealing with measurement data.

3) GAs simultaneously explore many points in the search space and combine knowledge. They are thus less likely to get stuck at a local optimum. It can be shown that GAs with certain properties will converge to a global optimum [1]. These advantages are gained at the expense of higher CPU time.

The procedure for running a simple GA is shown in Algorithm 2.1. Solutions to the problem are encoded as binary strings. A population of these strings is created randomly in the first step. The fitness value calculation (line#2&11) associates each string with a real value indicating the string's performance. This value is then used in line#6 to select two high fitness strings. The two selected strings are then subjected to the crossover and mutation operations.

Table 2.1 A comparison between conventional optimization techniques and genetic algorithms.

Property	Conventional Optimization Techniques	Genetic Algorithms
Applicability	Applicable to a specific problem domain.	Applicable to variety of problem domains.
Number of points being simultaneously searched	Single point	Multiple points
Transition from current point to the next	Deterministic	Probabilistic
Prerequisites	Auxiliary knowledge such as gradient vectors.	An objective function to be optimized.
Initial guess	Provided by the user	Automatically generated by the algorithms
Flow of control	Mostly serial	Mostly parallel
Required CPU time	Small amount	Large amount
Quality of the result	Local optimum, depends on initial guess	Global optimum is more probable

```
SimpleGeneticAlgorithm
1 Create an initial random population.
2 Calculate the fitness value of each string.
3 Repeat until termination criteria is satisfied.
4    newPop ← {}.
5    Repeat while |newPop| < the desired population size.
6       Select two parents from the current population.
7       Crossover the two parents with probability p_c to obtain
        two new offspring.
8       Mutate each bit of the two offspring with probability p_m.
9       Put the two offspring into newPop.
10   Replace the current population with newPop
11   Calculate the fitness value of each string.
```

Algorithm 2.1 A Simple Genetic Algorithm

Selection is the main engine that drives the GA to the solution; without it, GAs are merely random search techniques. There are many schemes that can be used for selection operation. Associated with each selection scheme is a reproduction rate R_i, which is defined as the expected number of times that a string i whose fitness value f_i will be selected in a population of size N. The traditional GAs selection scheme proportionally assigns the string's probability of being selected on a roulette wheel and spins the wheel each time a string is needed. This scheme is called a fitness proportionate selection. The reproduction rate of the scheme is given by $R_i = \frac{f_i}{\bar{f}}$, where $\bar{f}$ is the average of the whole population. Because the population size is finite, the fitness proportionate scheme may produce sampling error- the number of times that the string i is actually selected may differ greatly from its expected value. Even though the effect of sampling error on GAs' performance is unknown, the result would be more predictable if the sampling error were eliminated. A stochastic universal selection (SUS) can be used for this purpose. SUS uses N equally spaced markers on the same roulette wheel instead of just one marker; therefore it needs to spin just once and N strings are selected simultaneously. SUS has a desirable property that the actual number of times that the string i will be selected is either $\lfloor R_i \rfloor$ or $\lceil R_i \rceil$. In a stochastic remainder sampling (SRS), a string is deterministically selected $\lfloor R_i \rfloor$ times. The fractional part of R_i is placed on the wheel as in fitness proportionate selection. All of these selection schemes are classified as proportional selections because R_i is proportional to f_i. One drawback of proportional selections is that they can lead to premature convergence: a situation in which a superfit individual rapidly spreads its genetic material in the population. This population can converge to an undesirable local optimum. Many modifications to the objective function, such as translation, scaling, and sigma truncation have been used to prevent premature convergence [2]-[5]. Some selection schemes do not base R_i directly on the fitness value, but on the rank of i. These schemes are called ordinal selection. Examples of ordinal selection schemes are linear ranking, exponential ranking, truncation, and tournament selection. In addition to help reducing the premature convergence effect (without modifying the objective function), ordinal selections are scale and translation invariant, i.e., the fitness values can be multiplied or added by a constant and the value of R_i will not change. This property simplifies the analysis of the selection method. In linear ranking, the value of R_i is given by $R_i = \eta^- + (\eta^+ - \eta^-)\frac{k_i - 1}{N-1}$, where k_i is the rank of i (higher-ranking individuals have higher fitness values), and η^-, η^+ are the minimum and maximum number of times for an individual to be selected respectively. Because $\eta^- \geq 0$ and N is a constant, a simple calculation shows that $\eta^+ + \eta^- = 2$. In exponential ranking,

$R_i = N \frac{c^{N-k_i}}{\sum_{j=1}^{N} c^{N-k_j}}$ where $0<c<1$ specifies the degree of exponentiality. A value of c closer to 0 allocates a larger R_i to higher-ranking individuals. In truncation selection, only the best T portion of the population is selected with equal probability. Thus the reproduction rate for the truncation selection is $R_i = \begin{cases} \frac{1}{T} & \text{if } k_i > T.N \\ 0 & \text{otherwise} \end{cases}$

Finally, in a tournament selection, t individuals are randomly selected to compete in a "tournament". Only the winner is selected as a parent. The tournament must therefore be held N times to select N individuals. From the analysis of [6], the reproduction rate of a tournament selection for tournament size t is $R_i = t\left(\frac{k_i}{N}\right)^{t-1}$. Tournament selection needs no global information about the population (such as the average and the rank). This makes it attractive for distributed systems where communication overhead should be minimized.

The next step in running GAs is crossover, which is a process that involves swapping portions of strings. Each time the crossover operation takes place, two selected strings from the mating pool are combined. A position along the selected string is selected at random. All binary digits following the position are swapped with the other string. The result is two new strings that move on to the next generation. Fig. 2.1 illustrates this operation, assuming that string A and string B have already been chosen. This simple crossover operation is called a one-point crossover. Other types of crossover include two-point, multi-point, uniform, partially-matched, order-based, and string-of-change crossover [5]. In our work, multi-point crossover, i.e., one point for each parameter of the string has been chosen as the crossover operation.

Mutation follows crossover and protects against permanent loss of useful genetic information. The operator works by toggling each bit in the string with probability ρ_m as shown in Fig. 2.2.

As the algorithm proceeds from one generation to the next, there is no guarantee that the best individual in the next population will be better than the current one. It can actually get worse due to genetic altering operations. An elitist strategy guarantees that the best individual of the next generation will not be worse than the current one by automatically transferring the current best to the next generation. This strategy also has the added benefit of guaranteeing a global convergence [1].

The GA shown in Algorithm 2.1 is called "generational" because the whole current population is replaced by the new population at once. In a "steady state" GA, only a few individuals are to be replaced at a time. A selection procedure in steady state GA therefore needs to select not only the "good" individuals but also the "bad" individuals to be removed from the population. In general, the term "generation gap" which assumes the value in the range [0,1], specifies the population portion to be generated anew (and therefore to be replaced) in each time step.

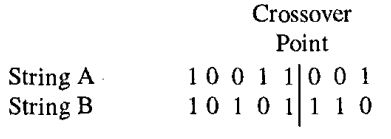

Fig. 2.1 The crossover operation of genetic algorithms

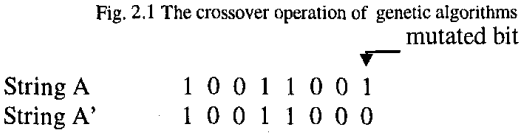

Fig. 2.2 The mutation operation of genetic algorithms

2.2 Genetic Programming

Genetic Programming(GP) uses the same "survival of the fittest" strategy as GAs do. However, GP is capable of evolving complex solutions with unknown size or structure by using a special data structure- a computer program. A computer program can be represented by a tree whose internal and external nodes are defined according to the problem being solved. A set of symbols that represent internal nodes of the tree is called a function set, and a set of symbols that represents external nodes of the tree is called a terminal set. To be able to evolve a given solution, the terminal and function sets must be selected so that they satisfy the closure and sufficiency properties. The closure property ensures that the generated trees are valid potential solutions. The sufficiency property ensures that the solution can be generated from the provided sets of symbols. As an example, suppose the problem is to rediscover Kepler's third law: $P_i = \sqrt{D_i^3}$,where D_i is the distance from planet i to the sun and P_i is the period of that planet. We can let the function set be {+, -, *, /, ^} and the terminal set be $\{D_i, i=1..9\}$. The solution that we are looking for is a tree that is equivalent to the tree in Fig. 2.3.

The procedure for running a GP technique is similar to that of running GAs described in the previous section. Some minor modifications are needed to handle the tree data structure. In the initialization step, a random population of these trees is generated. The size of the tree can grow indefinitely, so it is necessary to impose a bound on the size of the trees. The crossover operation of GP works by selecting a random node (either internal or external) in each parent tree. These two nodes represent the roots of the subtrees to be crossed over. The resulting offspring may greatly differ from their parents. For example in Fig. 2.4, two 3-node parents produce offspring with 5 nodes and 1 node respectively.

The mutation operation in GP works by selecting a random node in the tree and replacing that node with a new randomly generated subtree. Fig. 2.5 shows an example of a GP mutation operation.

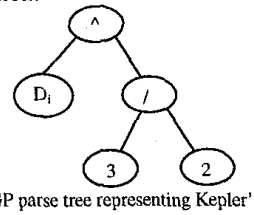

Fig. 2.3 A GP parse tree representing Kepler's 3rd law.

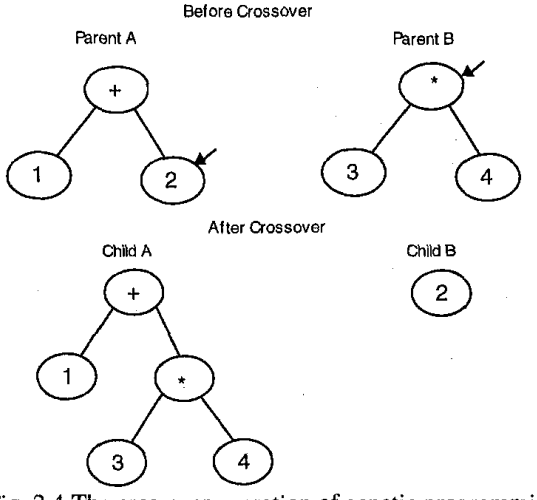

Fig. 2.4 The crossover operation of genetic programming.

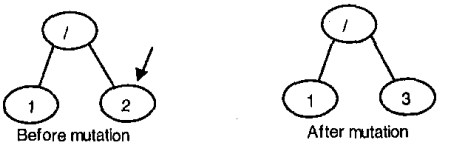

Fig. 2.5 The mutation operation of genetic programming

Other unique genetic operations of GP include editing and encapsulation. These operations mimic features that are available in modern high level languages. Editing simplifies the tree and may replace the tree with a smaller equivalent tree. Encapsulation generates parametered subtrees that can be repeatedly used by the main tree. The subtrees and the main tree can be dynamically evolved simultaneously.

Due to its flexibility, GP has been used for solving a variety of problems. It can be shown [7] that GP which includes an ability to store and retrieve data to/from memory is Turing complete, i.e., any computable task can be computed by using GP to evolve an algorithm for it. Furthermore, computer programs that produce results better than human performance have recently been evolved [8, 9].

GAs and GP share several similarities. Both of them are weak methods of problem solving (that is, they are not tied to any specific problem domain). They are also stochastic, population-based, and driven by the law of natural selection. However, the size of GAs search space is finite, i.e., 2^l for a binary string of length l. The size of GP search space can be arbitrarily large with many possible duplications of the same point. It is expected that GAs will outperform GP on problems where the search space is well defined, such as the parameter optimization problems being investigated here. However, this may not always be true as will be shown in this paper.

3. Motor Parameter Determination Problem

An induction motor parameter determination problem can be modeled by using an approximate equivalence circuit, an exact equivalent circuit, or a deep bar circuit model [13,14]. The parameters are calculated using the GA or GP and the torques are calculated using the circuit equations. The errors between the input motor torques and the calculated torques is an indication of the ability of the GA or GP to determine suitably accurate parameters. The input torques are full load, locked rotor, and breakdown torques obtained from real manufacturers' data.

In the formulation presented here, the motor parameters are assumed constant, largely because the intention is to use the parameters for system level studies where extreme precision is unnecessary. Higher accuracy may be obtained by allowing the leakage reactances to vary as a function of the current. The technique used here is therefore different from other techniques of parameter determination given by [15-17]. In [15,16], maximum likelihood estimators are used to identify equivalent circuit models, while [17] shows the problem associated with a graphically oriented approach.

3.1.1 Formulation Using the Approximate Equivalent Circuit.

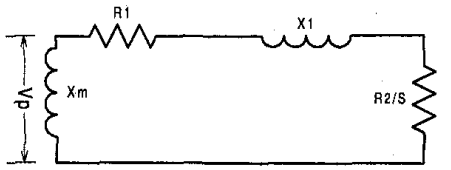

Fig. 3.1 An Approximate Equivalent Circuit

The function to be maximized in an approximate equivalence circuit is:

$$Fitness = \frac{100}{100 + |F_1| + |F_2| + |F_3|}$$

where

$$F_1 = 100 \frac{\dfrac{V^2 R_2}{S\left[\omega_s\left(R_1 + \dfrac{R_2}{S}\right)^2 + X_1^2\right]} - T_{fl}}{T_{fl}}$$

$$F_2 = 100 \frac{\dfrac{V^2 R_2}{\omega_s\left(R_1 + R_2\right)^2 + X_1^2} - T_{lr}}{T_{lr}}$$

$$F_3 = 100 \frac{\dfrac{V^2}{2\omega_s\left(R_1 + \sqrt{R_1^2 + X_1^2}\right)} - T_{bd}}{T_{bd}}$$

T_{fl}, T_{lr} and T_{bd} are the known values of full load torque, lock rotor torque, and breakdown torques respectively.

3.1.2 GA Implementation

Each parameter is encoded as a 14 bit unsigned binary number, together forming one 42 bit string as shown in Fig. 3.2. For each parameter in the string, 00...0 represents 0.0 ohm and 11...1 represents 100.0 Ohm. All intermediate values are linearly scaled between these two end points.

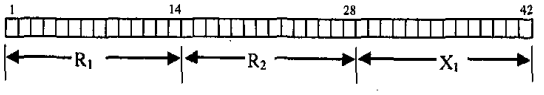

Fig. 3.2 The genetic algorithm string for an approximate circuit model.

3.1.3 GP Implementation

A terminal set for this problem is chosen as the set of random floating-point numbers in the range (0..10). The function set is {+, -, *, /, *asqrt*, *root*} where *asqrt* takes one floating-point parameter and returns the square root of the absolute value of the parameter, and *root* is a reserved symbol to act as the root of the tree that joins together all of its arguments. There is only one root for each tree and it is not subjected to genetic operations. In this model, *root* has 3

arguments; each argument corresponds to the value of R_1, R_2 and X_1 respectively. The division operator (/) performs a floating-point division and returns 1.0 if the divisor is 0.0.

3.2.1 Formulation Using the Exact Equivalent Circuit Model

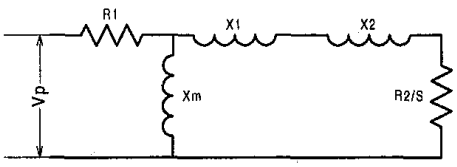

Fig. 3.3 An Exact Equivalent Circuit

In this model, the independent variables (R_1, R_2, X_1, X_2 and X_m) are determined from the equations:

$$fitness = \frac{100}{100+|F_1|+|F_2|+|F_3|+|F_4|}$$

where

$$F_1 = 100 - \frac{\dfrac{V_{th}^2 R_2}{\omega_s S\left[\left(R_{th}+\dfrac{R_2}{S}\right)^2+\left(X_{th}+X_2\right)^2\right]} - T_{fl}}{T_{fl}}$$

$$F_2 = 100 - \frac{\dfrac{V_{th}^2}{\omega_s\left(R_{th}+\sqrt{R_{th}^2+X_{th}^2+2X_{th}X_2+X_2^2}\right)} - T_{bd}}{T_{bd}}$$

$$F_3 = 100 - \frac{\dfrac{V_{th}^2 R_2}{\omega_s\left(\left(R_{th}+R_2\right)^2+\left(X_{th}+X_2\right)^2\right)} - T_{lr}}{T_{lr}}$$

$$F_4 = 100 - \frac{\cos\left(\arctan\left(\dfrac{X_{th}+X_2}{R_{th}+\dfrac{R_2}{s}}\right)\right) - pf}{pf}$$

$$R_{th} = \frac{X_m^2 R_1}{R_1^2+\left(X_1+X_m\right)^2}$$

$$X_{th} = \frac{X_m(R_1^2+X_1^2)+X_1X_m^2}{R_1^2+\left(X_1+X_m\right)^2}$$

$$V_{th} = \frac{V_p X_m}{\sqrt{R_1^2+(X_1+Xm)^2}}$$

3.2.2 GA Implementation

A 5 by 14 = 70 bit string is used to represent the value of the parameters. The binary value of each parameter is linearly mapped to the range [0,100].

3.2.3 GP Implementation

The same terminal and function sets are used as those in the approximate model, except that the *root* function has 5 parameters representing R_1, R_2, X_1, X_2 and X_m.

3.3.1 Formulation Using the Deep Bar Circuit Model

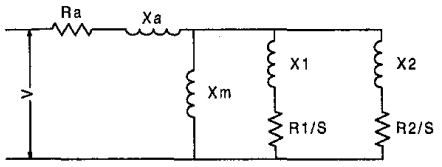

Fig. 3.4 A Deep Bar Circuit

In this model, the parameters R_1, R_2, R_a, X_1, X_2, X_a and X_m are to be determined from the following equations:

$$fitness = \frac{100}{100+|F_1|+|F_2|+|F_3|+|F_4|+|F_5|}$$

where

$$F_1 = \frac{\dfrac{3V^2|Y_a|^2\left[R_1|Y_1|^2+R_2|Y_2|^2\right]}{S\omega_s|Y_a+Y_m+Y_1+Y_2|^2} - T_{fl}}{T_{fl}}100$$

$$F_2 = \frac{\dfrac{3V^2|Y_a|^2\left[R_1|Y_1|^2+R_2|Y_2|^2\right]}{\omega_s|Y_a+Y_m+Y_1+Y_2|^2} - T_{lr}}{T_{lr}}100$$

$$F_3 = \frac{\dfrac{3V^2|Y_a|^2\left[R_1|Y_1|^2+R_2|Y_2|^2\right]}{S_{\max}\omega_s|Y_a+Y_m+Y_1+Y_2|^2} - T_{bd}}{T_{bd}}100$$

$$F_4 = \frac{\dfrac{(\Re(Y_1)+\Re(Y_2))|Y_a|+R_a|Y_a||Y_m+Y_1+Y_2|^2}{|Y_m+Y_1+Y_2||Y_a+Y_m+Y_1+Y_2|} - pf}{pf}100$$

$$F5 = \frac{\dfrac{V|Y_a||Y_m+Y_1+Y_2|}{|Y_a+Y_m+Y_1+Y_2|} - I_{fl}}{I_{fl}}100$$

$$Y_a = \frac{1}{R_a+jX_a} \qquad Y_m = -\frac{1}{jX_m}$$

$$Y_1 = \frac{1}{\frac{R_1}{S}+jX_1} \qquad Y_2 = \frac{1}{R_2+jX_2}$$

$S_{max} = S$ such that $\dfrac{3V^2|Y_a|^2\left[R_1|Y_1|^2+R_2|Y_2|^2\right]}{S\omega_s|Y_a+Y_m+Y_1+Y_2|^2}$ is maximized

3.3.2 GA Implementation

A 7 by 14 = 98 bit string is used to represent the value of the parameters. The binary value of each parameter is linearly mapped to [0..100].

3.3.3 GP Implementation

The same terminal and function sets are used as those in the approximate model, excepts that the *root* function has seven parameters representing R_1, R_2, R_a, X_1, X_2, X_a and X_m.

4. Results

The performance of GA and GP can be affected by numerical values of constants needed in the implementation, for example, the mutation rate. Wherever appropriate, the same values of the constants are used. Table 4.1 shows the value of each parameter that was used in this paper.

Three motors shown in Table 4.2 were tested using different circuit models and techniques. Table 4.3 compares the performance of GAs and GP on the average percentage torque errors of the final results for each model. Each entry in the table is an average of the best value obtained in 10 runs. Fig.4.1 shows typical percent torque errors as a function of the number of evaluations during the run of GA and GP.

The progress shown in Fig. 4.1 indicates that both the GA and the GP methods converge rapidly to low values of overall torque errors. Table 4.3 has additional details of the full load, locked rotor, and break down torques using 3 different motor sizes. The use of the approximate equivalent circuit prevents the GA or the GP from determining the motor parameters accurately. For example, the 5 HP machine has a 20% error in calculating the full load torque using the approximate equivalent circuit, which reduces to 3% and 6% when using the exact and deep bar models. In some cases, the GA performed better than the GP (for example, motor#1 with the approximate equivalent circuit) and in other cases, the GP performed better (for example, motor#3 using the exact equivalent circuit). However, the magnitudes of the torque errors using the GP method were in general smaller than those using the GA method. Generally, the deep bar model gave better results thus reaffirming the need to use a more accurate model in the motor parameter determination problem, particularly covering a wide speed range.

This technique has been applied to motor parameter determination for system studies where high accuracy is generally not required and no knowledge of the parameters are available a priori. Parameters variations through saturation can be included but would require detailed motor data which is generally not available in system level studies.

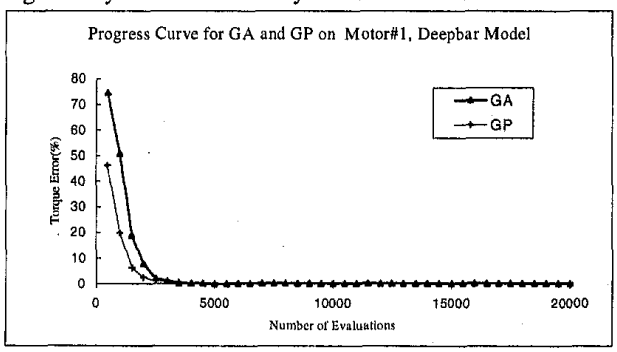

Fig.4.1 A typical progress curve of GA&GP on motor determination problem

Table 4.1 The value of each parameter for GA and GP used in the experiment.

Parameter	Value used by GA	Value used by GP
Population Size	500	500
Number of Evaluations	25000	25000
Evolution Model	Steady State	Steady State
Selection Method	Tournament Selection	Tournament Selection
Tournament Size	3	3
Crossover Type	Multi-point	1 point
Crossover Rate	0.8	0.8
Mutation Rate	0.001(per bit)	0.001(per tree)
Use elitism	Yes	Yes
Max. tree height	N/A	7
Max. tree size	N/A	512 nodes

Table 4.2. The input motor parameters used in the experiment.

Motor#	HP	Voltage	RPM	T_{fl}	T_{lr}	T_{bd}
1	5	200	1760	14.9	33.5	50.2
2	50	460	1765	149	300	408
3	250	460	1760	738	1040	2051

Table 4.3. The average of the best torque errors obtained from GA and GP.

Motor#	Model	Tq.	Act.	GA		GP	
			Value	Calc	%Err	Calc	%Err
1	Appx	Tfl	14.9	11.9	-20.2	14.5	-2.9
		Tlr	33.5	32.2	-3.9	31.2	-7.0
		Tbd	50.2	52.5	4.6	53.1	5.9
	Exact	Tfl	14.9	14.5	-2.7	14.4	-3.2
		Tlr	33.5	37.2	11.1	37.7	12.4
		Tbd	50.2	48.6	-3.2	47.1	-6.2
	Deepbar	Tfl	14.9	15.8	6.4	14.9	0.0
		Tlr	33.5	33.6	0.2	33.5	0.1
		Tbd	50.2	51.1	1.8	52.2	4.0
2	Appx	Tfl	149	109	-26.9	138.6	-7
		Tlr	300.0	281.0	-6.4	284.8	-5.1
		Tbd	408.0	453.5	11.2	493.1	20.9
	Exact	Tfl	149.0	124.6	-16.4	141.4	-5.1
		Tlr	300.0	350.2	16.7	369.8	23.3
		Tbd	408.0	394.6	-3.3	377.2	-7.5
	Deepbar	Tfl	149.0	164.6	10.5	149.8	0.6
		Tlr	300.0	312.8	4.3	301.2	0.4
		Tbd	408.0	452.8	11.0	440.8	8.0
3	Appx	Tfl	738.0	479.0	-35.1	690.1	-6.5
		Tlr	1040.0	896.4	-13.8	986.2	-5.2
		Tbd	2051.0	1998.5	-2.6	2077.3	1.3
	Exact	Tfl	738.0	531.2	-28.0	713.0	-3.4
		Tlr	1040.0	1077.8	3.6	1105.0	6.3
		Tbd	2051.0	2127.7	3.7	2032.1	-0.9
	Deepbar	Tfl	738.0	902.0	22.2	761.8	3.2
		Tlr	1040.0	1051.3	1.1	1043.2	0.3
		Tbd	2051.0	2356.6	14.9	2174.9	6.0

5. Conclusion

This paper has presented the fundamental concepts of genetic algorithms and genetic programming, and shown how a parameter determination problem in induction machines can be formulated to allow its solution using those techniques. Three different motor sizes using three different equivalent circuits were considered. The GA produced better results in some instances and the GP produced better results in other instances. These motor parameters can be used in system-level studies like the calculation of reclosing transients for motor protection.

REFERENCES

[1] Gunter Rudolph, "Convergence Analysis of Canonical Genetic Algorithms", *IEEE Trans on Neural Networks*, Vol. 5 No. 1, January 1994. pp96-101.

[2] David E. Goldberg, *Genetic Algorithms in Search, Optimization and machine Learning*, Addison-Wesley Publishing Company, Massachusetts, 1989.

[3] Lawrence Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, NY 1991.

[4] Melany Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1996.

[5] Reeves, C.R., ed., *Modern Heuristic Techniques for Combinatorial Problems*, Halsted Press, 1993.

[6] Tobias Blickle and Lothar Thiele, "A Comparison of Selection Schemes used in Genetic Algorithms", TIK-Report No. 11, December 1995, Computer Engineering and Communication Network Lab, Swiss Federal Institute of Technology, Switzerland.

[7] Astro Teller, "Turing Completeness in the Language of Genetic Programming with Indexed Memory", *IEEE Conference on Evolutionary Computation - Proceedings v/1 1994*. IEEE, Piscataway, NJ, 94TH0650-2. p136-141.

[8] Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. "Four Problems for which a Computer Program Evolved by Genetic Programming is Competitive with Human Performance". *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*. IEEE Press. p1-10.

[9] David Andre, Forrest H. Bennette III and John R. Koza, "Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem". *Genetic Programming, Proceedings of the First Annual Conference*. pp3-11.

[10] John R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, The MIT Press, 1992.

[11] John R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, The MIT Press, 1994.

[12] Zbigniew Michalewicz, *Genetic Algorithms + Data Stucture = Evolution Programs*, Springer-Verlag , 1992.

[13] B.K. Johnson and J.R. Willis, "Tailoring Induction Motor Analytical Models to Fit Known Motor Performance Characteristics and Satisfy Particular Study Needs", *IEEE Transactions on Power Systems*, vol. 6, no. 3, August 1991.

[14] R. Nolan, P. Pillay, T. Haque, "Application of Genetic Algorithms to Motor Parameter Determination," in *Proceedings of 1994 IEEE-IAS Conference*, Denver, October 1994, pp. 47-54.

[15] S. Horning, A. Keyhani, I.Kamwa, "On-Line Evaluation of Round Rotor Synchronous Machines Parameter Set Estimated from Standstill Time-Domain Data", IEEE Transactions on Energy Conversion, Vol.12, No.4, December 1997, pp. 289-296.

[16] A. Keyhani, H. Tsai "Identification of High-Order Synchronous Generator Models from SSFR Test Data", IEEE Transactions on Energy Conversion, Vol.9, No.3, September 1994. pp. 593-600.

[17] I. Kamwa, P. Viarouge, R. Mahfoudi "Phenomenological Models of Large Synchronous Machines from Short–Circuit Tests During Commissioning-A Classical/Modern Approach", IEEE Transactions on Energy Conversion, Vol.9, No.1, March 1994. pp. 85-97.

LIST OF SYMBOLS

The symbols used in this paper are defined below.

- l The length of a genetic algorithm string.
- N The population size.
- k_i The rank of an individual i. Better individuals have higher ranks.
- R_i The reproduction rate of an individual i.
- f_i The fitness value of an individual i.
- $\overline{f}$ The average fitness value of the population.
- η^-, η^+ The minimum and maximum reproduction rates for a genetic algorithm using a linear ranking selection.
- c The degree of exponentiality for an exponential ranking selection.
- T The truncation size in truncation selection.
- t The tournament size in tournament selection.
- T_{fl}, T_{bd}, T_{lr} The full load, break down, and lock rotor torques.
- I_{fl} The full load current.
- pf The power factor.
- S The slip factor.
- ω_s The motor's angular velocity.
- $\boldsymbol{Y}$ The admittance (complex-valued).
- $\Re$ A function that returns the real part of the given complex number.

BIOGRAPHY

Phaderm Nangsue received B.S. degrees in Computer Science Engineering, and in Mechanical Engineering from Norwich University, VT in 1990, and the M.S.degree in Computer Engineering from Clarkson University in 1992. He is currently pursuing the Ph.D degree in Computer Engineering at Clarkson University under the supervisory of Dr. Susan E. Conry. His research interests include artificial intelligence, genetic algorithms, and distributed problem solving.

Pragasen Pillay (S'84-M'87-SM'92) received the Bachelor's degree from the University of Durban-Westville in South Africa in 1981, the Master's degree from the University of Natal in South Africa in 1983 and the Ph.D from Virginia Polytechnic Institute & State University in 1987, while funded by a Fulbright Scholarship. From January 1988 to August 1990 he was with the University of Newcastle upon Tyne in England. From August 1990 to August 1995 he was at the University of New Orleans. Currently he is with Clarkson University, Potsdam, NY 13699 where he is a Professor in the Department of Electrical & Computer Engineering and holds the J. Newell Distinguished Professorship in Engineering. He is a senior member of the IEEE and a member of the Power Engineering, Industry Applications, Industrial Electronics and Power Electronics Societies. He is a member of the Electric Machines Committee, Vice-chairman (programs) of the Industrial Drives Committee and Vice-chairman of the Continuing Education Subcommittee within the Industry Applications Society. He is a member of the IEE, England and a Chartered Electrical Engineer. He has organized and taught short courses in electric drives at the Annual Meeting of the Industry Applications Society. His research and teaching interests are in modeling, design and control of electric motors and drives.

Susan E. Conry (M'77) received the bachelor's degree in mathematics in 1971, and the M.S. and Ph.D. degrees in electrical engineering in 1973 and 1975, all from Rice University, Houston, TX.

She is an Associate Professor in the Department of Electrical and Computer Engineering at Clarkson University. She has been interested in various aspects of parallel and distributed computation for many years. Most recently, her research has been largely concerned with distributed problem solving and distributed planning systems.