Ensemble of Genetic Programming Models for Designing Reactive Power Controllers

Crina Grosan and Ajith Abraham*
Department of Computer Science, Babes-Bolyai University,
Cluj-Napoca, Romania, cgrosan@cs.ubbcluj.ro
School of Computer Science and Engineering,
Chung-Ang University, South Korea, ajith.abraham@ieee.org

Abstract

In this paper, we present an ensemble combination of two genetic programming models namely Linear Genetic Programming (LGP) and Multi Expression Programming (MEP). The proposed model is designed to assist the conventional power control systems with added intelligence. For on-line control, voltage and current are fed into the network after preprocessing and standardization. The model was trained with a 24-hour load demand pattern and performance of the proposed method is evaluated by comparing the test results with the actual expected values. For performance comparison purposes, we also used an artificial neural network trained by a backpropagation algorithm. Test results reveal that the proposed ensemble method performed better than the individual GP approaches and artificial neural network in terms of accuracy and computational requirements.

1 Introduction

A wide variety of real time power monitoring /control systems are helping the electrical power consumers [1][6]. For the monitoring system to be more intelligent, we propose the use of an ensemble combination of two genetic programming models namely Linear Genetic Programming (LGP) and Multi Expression Programming (MEP) for predicting the trend of reactive power demand. By predicting the reactive power demand it is possible to automate the control of reactive power load and better utilization of volt-amperes (VA) inflow. Efficient usage of the VA loading will not only improve the overall grid condition but also reduce the consumer's industrial tariffs. Depending on the predicted reactive power demand, power factor corrective measures could be turned on or off to control the VA inflow into the plant. This prediction system will be extremely useful for automated control of power inflow, especially in the countries where there are limitations on the usage of consumer's peak VA maximum demand.

2 Importance of Reactive Power Control

The ratio of active power (P) measured in watts to the apparent power (S) in volt-amperes is termed the power factor. It has become a normal practice to say that the power factor is lagging when the current lags the supply voltage and leading when the current leads the supply voltage. This means that the supply voltage is regarded as the reference quantity. A majority of loads served by a power utility draw current at a lagging power factor. When the power factor of the load is unity, active power equals apparent power ($P = S$). But, when the power factor of the load is less than unity, say 0.6, the power utilized is only 60%. This means that 40% of the apparent power is being utilized to supply the reactive power, VAR, demand of the system. It is therefore clear that the higher the power factor of the load, the greater the utilization of the apparent power [4]. For the generating and transmission stations, lower the power factor the larger must be the size of the source to generate that power, and greater must be the cross-sectional area of the conductor to transmit it. In other words, the greater is the cost of generation and transmission of the power. Moreover, lower power factor will also increase the I^2R (I denotes current) losses in lines/equipment as well as result in poor voltage regulation [3][5][8].

We considered a heavy automobile industry for studying the load demand patterns. The plant works on 3 shifts of 8 hours duration each. The difference between the apparent and active power contributes for the reactive power. Observed data for a 24 hour period shows that the maximum and minimum VAR requirements are 2.96 MVAR and 0.014 MVAR, respectively. If suitable power factor compensation was made when the reactive power demand was increasing, the plant might not have drawn much apparent power from the grid. The task is to predict the upward and downward trend of the reactive power demand and provide required power factor compensation. Load flow analysis of the captioned plant reveals that the demand patterns are very similar every day (as long as the production of automobiles remains fixed). This paper presents an ensemble combination of two Genetic Programming (GP) models and the performance is compared with an artificial neural network trained by a backpropagation algorithm. The proposed models were trained on the data taken at every minute for a 24-hour period to predict the reactive power demand parameters, and tested to evaluate the prediction accuracy.

3 Hybrid Modeling of Intelligent Paradigms

3.1 Linear Genetic Programming (LGP)

Linear genetic programming is a variant of the GP technique that acts on linear genomes [11]. Its main

Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS'05)
0-7695-2457-5/05 $20.00 © 2005 IEEE

characteristics in comparison to tree-based GP lies in that the evolvable units are not the expressions of a functional programming language (like LISP), but the programs of an imperative language (like c/c ++). An alternate approach is to evolve a computer program at the machine code level, using lower level representations for the individuals. This can tremendously hasten the evolution process as, no matter how an individual is initially represented, finally it always has to be represented as a piece of machine code, as fitness evaluation requires physical execution of the individuals. The basic unit of evolution here is a native machine code instruction that runs on the floating-point processor unit (FPU). Since different instructions may have different sizes, here instructions are clubbed up together to form instruction blocks of 32 bits each. The instruction blocks hold one or more native machine code instructions, depending on the sizes of the instructions. A crossover point can occur only between instructions and is prohibited from occurring within an instruction. However the mutation operation does not have any such restriction. LGP uses a specific linear representation of computer programs. Instead of the tree-based GP expressions of a functional programming language (like *LISP*) programs of an imperative language (like *C*) are evolved. A LGP individual is represented by a variable-length sequence of simple *C* language instructions. Instructions operate on one or two indexed variables (registers) *r*, or on constants *c* from predefined sets. The result is assigned to a destination register, for example, *ri = rj* c*. Here is an example LGP program:

```
void LGP(double v[8])
[0] = v[5] + 73;
v[7] = v[3] – 59;
if (v[1] > 0)
if (v[5] > 21)
v[4] = v[2] . v[1];
v[2] = v[5] + v[4];
v[6] = v[7] . 25;
v[6] = v[4] – 4;
v[1] = sin(v[6]);
if (v[0] > v[1])
v[3] = v[5] . v[5];
v[7] = v[6] . 2;
v[5] = v[7] + 115;
if (v[1] <= v[6])
v[1] = sin(v[7]);
}
```

A LGP can be turned into a functional representation by successive replacements of variables starting with the last effective instruction. The maximum number of symbols in a LGP chromosome is 4 * Number of instructions.

Evolving programs in a low-level language allows us to run those programs directly on the computer processor, thus avoiding the need of an interpreter. In this way the computer program can be evolved very quickly. An important LGP parameter is the number of registers used by a chromosome. The number of registers is usually equal to the number of attributes of the problem. If the problem has only one attribute, it is impossible to obtain a complex expression such as the quartic polynomial. In that case we have to use several supplementary registers. The number of supplementary registers depends on the complexity of the expression being discovered. An inappropriate choice can have disastrous effects on the program being evolved. LGP uses a modified steady-state algorithm. The initial population is randomly generated. The following steps are repeated until a termination criterion is reached: Four individuals are randomly selected from the current population. The best two of them are considered the winners of the tournament and will act as parents. The parents are recombined and the offspring are mutated and then replace the losers of the tournament. We used a LGP technique that manipulates and evolves a program at the machine code level. The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. The population space has been subdivided into multiple subpopulation or demes. Migration of individuals among the subpopulations causes evolution of the entire population. It helps to maintain diversity in the population, as migration is restricted among the demes. Moreover, the tendency towards a bad local minimum in one deme can be countered by other demes with better search directions. The various LGP search parameters are the mutation frequency, crossover frequency and the reproduction frequency: The crossover operator acts by exchanging sequences of instructions between two tournament winners. Steady state genetic programming approach was used to manage the memory more effectively.

3.2. Multi Expression Programming (MEP)

MEP genes are (represented by) substrings of a variable length [9][10]. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of lower values than the position of the function itself in the chromosome. This representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme, the first symbol of the chromosome must be a terminal symbol. In this way, only syntactically correct programs (MEP individuals) are obtained. An example of chromosome using the sets F= {+, *} and T= {a, b, c, d} is given below:

1: *a*
2: *b*
3: + 1, 2
4: *c*
5: *d*

6: + 4, 5
7: * 3, 6
The maximum number of symbols in MEP chromosome is given by the formula:
Number_of_Symbols = (*n* + 1) * (*Number_of_Genes* – 1) + 1,
where *n* is the number of arguments of the function with the greatest number of arguments. The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments. The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.
The translation of a MEP chromosome into a computer program represents the phenotypic transcription of the MEP chromosomes. Phenotypic translation is obtained by parsing the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol.

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:
$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression: $E_3 = a + b$. Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression: $E_6 = c + d$. Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression: $E_7 = (a + b) * (c + d)$. E_7 is the expression encoded by the whole chromosome.
There is neither practical nor theoretical evidence that one of these expressions is better than the others. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number of genes). The chromosome described above encodes the following expressions:
$E_1 = a$,
$E_2 = b$,
$E_3 = a + b$,
$E_4 = c$,
$E_5 = d$,
$E_6 = c + d$,
$E_7 = (a + b) * (c + d)$.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming and are stored in a conventional manner.

Due to its multi expression representation, each MEP chromosome may be viewed as a forest of trees rather than as a single tree, which is the case of Genetic Programming.

Fitness assignment
As MEP chromosome encodes more than one problem solution, it is interesting to see how the fitness is assigned. The chromosome fitness is usually defined as the fitness of the best expression encoded by that chromosome. For instance, if we want to solve symbolic regression problems, the fitness of each sub-expression E_i may be computed using the formula:

$$f(E_i) = \sum_{k=1}^{n} \left| o_{k,i} - w_k \right|,$$

where $o_{k,i}$ is the result obtained by the expression E_i for the fitness case k and w_k is the targeted result for the fitness case k. In this case the fitness needs to be minimized. The fitness of an individual is set to be equal to the lowest fitness of the expressions encoded in the chromosome:

When we have to deal with other problems, we compute the fitness of each sub-expression encoded in the MEP chromosome. Thus, the fitness of the entire individual is supplied by the fitness of the best expression encoded in that chromosome.

3.3. Ensemble Modeling of LGP and MEP

Our goal is to optimize two error measures namely Root Mean Squared Error (RMSE) and Correlation Coefficient (CC):

$$RMSE = \sqrt{\sum_{i=1}^{N} \left| P_{actual,i} - P_{predicted,i} \right|}$$

$$CC = \frac{\sum_{i=1}^{N} P_{predicted,i}}{\sum_{i=1}^{N} P_{actual,i}}$$

The task is to have minimal value of RMSE and a maximum value for CC. The objective is to carefully construct the different GP models to achieve the best generalization performance. Test data is then passed through these individual models and the corresponding outputs are recorded. Suppose results obtained by LGP and MEP are a_n and b_n respectively and the corresponding desired value is x_n. The task is to combine a_n and b_n so as to get the best output value that maximizes the CC and minimizes the RMSE values.

We consider this problem as a multiobjective optimization problem in which we want to find solution of this form: ($coef_1$, $coef_2$), where $coef_1$, and $coef_2$ are real numbers between - 1 and 1, so as the resulting combination: $coef_1 * a_n + coef_2 * b_n$
would be close to the desired value x_n. This means, in fact, to find a solution so that to simultaneously optimize RMSE and CC. This problem is equivalent to find the Pareto solutions of a multiobjective optimization problem.

In our situation, the objectives are RMSE and CC. We use for this problem a well known Multiobjective Evolutionary Algorithm (MOEA) – Nondominated Sorting Genetic Algorithm II (NSGAII) [12]. A short description of this algorithm is given below.

3.3.1 Nondominated Sorting Genetic Algorithm II (NSGA II)

K. Deb et al. [12] suggested a fast elitist Nondominated Sorting Genetic Algorithm (NSGA II). In NSGA II, for each solution x the number of solutions that dominate solution x is calculated. The set of solutions dominated by x is also calculated. The first front (the current front) of the solutions that are nondominated is obtained.

Let us denote by S_i the set of solutions that are dominated by the solution x^i. For each solution x^i from the current front consider each solution x^q from the set S_i. The number of solutions that dominates x^q is reduced by one. The solutions which remain non-dominated after this reduction will form a separate list. This process continues using the newly identified front as the current front. Let $P(0)$ be the initial population of size N. An offspring population $Q(t)$ of size N is created from current population $P(t)$. Consider the combined population R(t) = P(t) $\cup$ Q(t).

Population $R(t)$ is ranked according to nondomination. The fronts F_1, F_2, ... are obtained. New population $P(t+1)$ is formed by considering individuals from the fronts F_1, F_2, ..., until the population size exceeds N. Solutions of the last allowed front are ranked according to a crowded comparison relation.

NSGA II uses a parameter (called *crowding distance*) for density estimation for each individual. Crowding distance of a solution x is the average side-length of the cube enclosing the point without including any other point in the population. Solutions of the last accepted front are ranked according to the crowded comparison distance. NSGA II works as follows. Initially a random population, which is sorted based on the nondomination, is created. Each solution is assigned a fitness equal to its nondomination level (1 is the best level). Binary tournament selection, recombination and mutation are used to create an offspring population. A combined population is formed from the parent and offspring population. The population is sorted according to the nondomination relation. The new parent population is formed by adding the solutions from the first front and the followings until exceed the population size. Crowding comparison procedure is used during the population reduction phase and in the tournament selection for deciding the winner.

3.4 Artificial Neural Network Model

Artificial Neural Networks (ANNs) have been developed as generalizations of mathematical models of biological nervous systems. A neural network is characterized by the network architecture, the connection strength between pairs of neurons (weights), node properties, and updating rules. The updating or learning rules control weights and/or states of the processing elements (neurons). The network is initially randomized to avoid imposing any of our own prejudices about an application on the network.

4. Experiment Setup, Analysis and Results

The experiment system consists of two stages: Model construction/network training and performance evaluation. A heavy automobile manufacturing plant was considered for the prediction of reactive power. All the training data were standardized before training. The input parameters considered are the Voltage (V) and Current (I). We randomly fluctuated the input parameter voltage (V) +/- 2.5% of the normal value to cater for worst conditions in the grid voltage regardless of the plant load. This also tests the learning ability of ANN during worst situations. Figures 1 and 2 show the input parameters V and I of the test data.

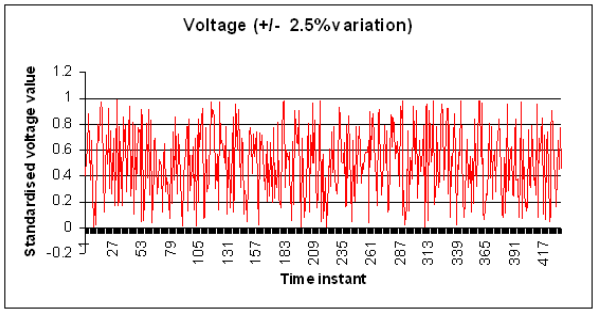

Figure 1. Test data- input voltage (+/- 30%)

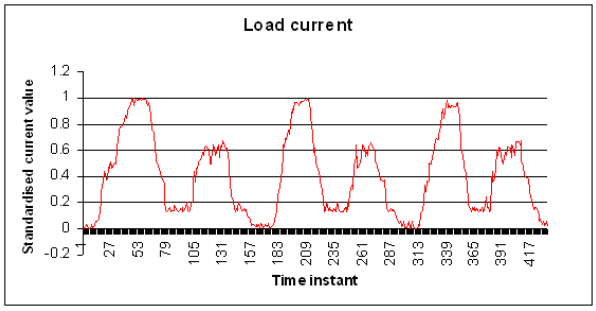

Figure 2. Test data- load current (amperes)

4.1 Parameter settings

For ANN, we used a feedforward network with 2 hidden layers in parallel, 2 input neurons corresponding to the input variables and 1 output neuron. The network was trained using 60% of the data and the remaining 40% data was used for testing and validation. Initial weights, learning rate and momentum used were 0.3, 0.1 and 0.1, respectively.

Proceedings of the Fifth International Conference on Hybrid Intelligent Systems (HIS'05)
0-7695-2457-5/05 $20.00 © 2005 IEEE

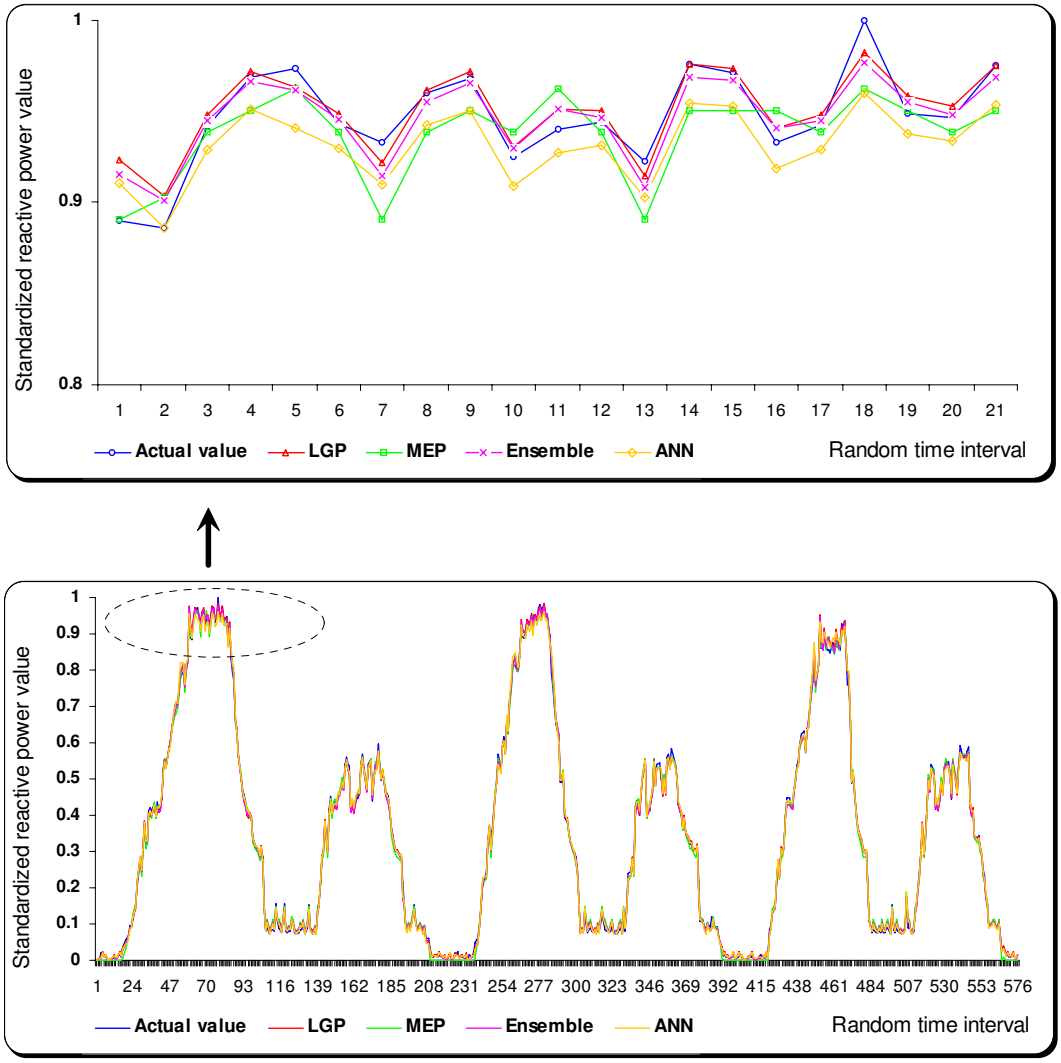

Figure 3. Comparison between ANN, MEP, LGP and ensemble

The training was terminated after 1500 epochs. Parameters used by MEP, LGP and Ensemble between LGP and MEP using NSGA II are depicted in Table 1, Table 2 and Table 3 respectively.

Table 1. Parameters used by MEP

Parameter	Value
Population size	50
Chromosome length	40
Number of generations	150
Crossover probability	0.9
Number of mutations/chromosome	3
Number of constants	10

Table 2. Parameters used by LGP

Parameter	Value
Population size	100
Mutation frequency	95%
Number of demes	10
Crossover frequency	50%
Number of constants	60

Table 3. Parameters used by Ensemble

Parameter	Value
Population size	250
Number of generations	500
Crossover probability	0.5

4.2 Comparison of results

Table 4 shows the comparative performance of ANN, MEP, LGP and ensemble between LGP and MEP for the reactive power prediction problem.

Table 4. Reactive power prediction performance

	ANN	MEP	LGP	Ensemble
RMSE	0.01210	0.0141	0.01104	0.0106
CC	0.9992	0.996	0.994	0.9999

From the experimental results, it is clear that the results obtained by the ensemble between GP techniques outperformed each of the individual techniques (LGP, MEP and neural network) in terms of performance time and error achieved. In Figure 3 the graphical comparison between ANN, MEP, LGP and Ensemble is presented.

5. Conclusions

This paper presented three techniques for the reactive power prediction problem. The ANN was clearly dominated by the GP techniques. We also combined the two GP techniques used so that to optimizes the error. The different GP techniques (LGP and MEP) were combined using an ensemble approach by an evolutionary multiobjective algorithm so as to simultaneously optimize the RMSE and CC. We evolved a set of coefficients in order to obtain an ensemble combination of the two techniques by applying a well known multiobjective evolutionary algorithm called NSGA II. Empirical results also illustrate that a combination of these techniques is very useful. The results obtained by an ensemble of these paradigms clearly outperform results obtained by each technique individually.

For this problem, we considered random values of input parameter voltage to test the learning ability of connectionist systems during worst conditions. The performance could have been even better if the observed rather than fluctuated values of voltage were used. Moreover, the considered connectionist models are very robust, capable of handling the noisy and approximate data that are typical in power systems, and therefore should be more reliable during worst conditions.

6. Acknowledgements

This research was supported by the International Joint Research Grant of the IITA (Institute of Information Technology Assessment) foreign professor invitation program of the MIC (Ministry of Information and Communication), Korea.

References

[1] Sawyer D., *Non-stop Monitoring, IEE Review*, Volume 45(3), May 1999.

[2] Nauk D., Klawonn F and Kruse R, *Foundations of Neuro Fuzzy Systems*, John Willey & Sons, 1997.

[3] Miller T.J.E., *Reactive Power Control in Electric Systems*, *Wiley – Interscience*, 1982.

[4] Dorf R.C., *The Electrical Engineering Handbook*, *CRC-IEEE press*, 1997.

[5] Cory B.J., Weedy B.M., *Electric Power Systems*, (4th Edition), John Wiley & Sons; 1998.

[6] Abraham A. and Nath B., Artificial Neural Networks for Intelligent Real Time Power Quality Monitoring Systems, First International Power & Energy Conference, INT-PEC'99, 1999.

[7] Sugeno M., *Industrial Applications of Fuzzy Control*, Elsevier Science Pub Co., 1985.

[8] Sheble G.B., *Reactive Power: Basics, Problems and Solutions*, IEEE Press, 1987.

[9] M. Oltean, C. Grosan. Evolving Evolutionary Algorithms using Multi Expression Programming, Proceedings of the 7th European Conference on Artificial Life, Dortmund, Germany, pp. 651-658, 2003.

[10] M. Oltean and C. Grosan. A Comparison of Several Linear GP Techniques. Complex Systems, Vol. 14, Nr. 4, pp. 285-313, 2003

[11] Banzhaf. W., Nordin. P., Keller. E. R., Francone F. D. Genetic Programming: An Introduction on The Automatic Evolution of Computer Programs and its Applications, Morgan Kaufmann Publishers, Inc., 1998.

[12] Deb K., Agrawal S., Pratab A., Meyarivan T., A fast elitist non-dominated sorting genetic algorithms for multiobjective optimization: NSGA II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.