A Linear Genetic Programming Approach for Modelling Electricity Demand Prediction in Victoria

Maumita Bhattacharya, Ajith Abraham, Baikunth Nath

School of Computing & Information Technology, Monash University,
Churchill, Victoria 3842, Australia
E-mail: {maumita.bhattacharya, ajith.abraham, b.nath}@infotech.monash.edu.au

Abstract

Genetic programming (GP), a relatively young and growing branch of evolutionary computation is gradually proving to be a promising method of modelling complex prediction and classification problems. This paper evaluates the suitability of a linear genetic programming (LGP) technique to predict electricity demand in the State of Victoria, Australia, while comparing its performance with two other popular soft computing techniques. The forecast accuracy is compared with the actual energy demand. To evaluate, we considered load demand patterns for ten consecutive months taken every 30 minutes for training the different prediction models. Test results show that while the linear genetic programming method delivered satisfactory results, the neuro fuzzy system performed best for this particular application problem, in terms of accuracy and computation time, as compared to LGP and neural networks.

Keywords: *Linear genetic programming, neuro-fuzzy, neural networks, forecasting, electricity demand.*

1. Introduction

In 1994 Victoria started the process of privatisation and restructuring electricity industry to generate competition. The objective was to promote a more flexible, cost-effective and efficient electricity industry with the aim of delivering cheaper electricity to business and the general community. Following success of this operation, Australia started the process of implementing a unified National Electricity Market in December 1998 [13]. More than 80 percent of electricity in the State comes from brown coal fired stations. Since the on/off states for such power generating stations have large time lags for start-ups, stability in its operations is important. The peak demand for electricity for the State at any time instant is about 6700 MWh. This demand is highly volatile on a day-to-day basis

and is being significantly affected by the Victorian weather conditions. Electricity is consumed as it is generated and is very often sold in advance of production. Electricity as a commodity has very different characteristics compared to a physical commodity. It cannot be stored. Therefore, to meet the electricity market demands a highly reliable supply and delivery system is required. Additionally, in order to gain a competitive advantage in this market through the spot-market pricing an accurate forecast of electricity demand at regular time intervals is essential. Until 1996, Victorian Power Exchange (VPX) the body responsible for the secure operations of the power system, generated electricity demand forecasts based on weather forecasts and historical demand patterns [8]. Our research is focused on developing more accurate and reliable forecasting models that improves current forecasting methods [2].

Genetic programming has been already successfully implemented to a variety of machine learning problems [1]. This paper investigates the suitability of linear genetic programming technique, for predicting 96 half-hourly (two days ahead) demands for electricity. Performance of the LGP technique has been compared with two popular soft computing forecasting models used in [2]. For developing the forecasting models we used the energy demand data for ten months period from 27th January to 30th November 1995 in the State of Victoria. We also made use of the associated data stating the minimum and maximum temperature of the day, time of day, season and the day of week. The forecasting models were trained using 3 randomly selected samples containing 20% of the data. To ascertain the forecasting accuracy the developed models were tested to predict the demand for a two days period 29-30 November 1995. The paper is organised as follows. In Section 2, we give a brief overview of the linear genetic programming, neuro-fuzzy system and artificial neural network techniques. Section 3 discusses the experimentation set-up, characteristics of the data and the forecasting performance of the various techniques. The analysis and the comparisons of the test results are summarised in section 4. Some conclusions are also provided towards the end.

2. Forecasting Models

A wide variety of forecasting methods are available to the management. A forecasting method may be treated as a machine-learning problem, where learning involves mapping of known inputs to known outputs and prediction based on learning. The evolution of soft computing techniques has increased the understanding of various aspects of the problem environment and consequently the predictability of many events. Soft computing models make use intelligent techniques including methods of neurocomputing, fuzzy logic, probabilistic computing, neuro-fuzzy computing, evolutionary algorithms and several hybrid techniques [11]. In contrast with the conventional AI techniques, which deal only with precision, certainty and rigor, connectionist models are able to exploit the tolerance for imprecision, uncertainty and are often very robust. Genetic programming tackles the problem of forecasting with the following added advantages:

i) The GP technique generates more transparent solution in the form of executable codes.

ii) It simultaneously evolves the structures and parameters to the solution.

iii) The problem of over fitting and memorization that often plagues the connectionist models can be minimized.

iv) Logically GP does not require any formal architecture selection as required in many connectionist models.

2.1 Genetic Programming

Of the different sub fields of evolutionary computation techniques, genetic programming is a comparatively young and growing research area. It can be treated as a machine learning technique that generates computer programs automatically. The novelty of genetic programming lies in its ability to abstract an underlying principle from a finite set of fitness cases and presenting them in the form of an algorithm or computer program to represent the simulation model [1] [4] [7].

Linear Genetic Programming

Linear genetic programming is a variant of the GP technique that acts on *linear genomes*. Its main 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 up 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. This concept was expanded to the AIMGP (*Automatic Induction of Machine code by Genetic Programming*) technique. In AIMGP, individuals are directly manipulated, as binary machine codes and executed without the use of an interpreter [1,6]. A general representation of the *linear genome* is illustrated in Figure 1. The linear genetic programming technique used for our current experiment is based on machine code level manipulation and evaluation of programs [6, 3]. 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.

The *linear GP algorithm* used in the current context is as described below:

1. **Initialization.** Create and initialize a population of randomly generated programs.
2. **Tournament Contest.** Randomly select four programs from the population. Based on ì fitnessî , two ì winnersî and two ì losersî are selected.

3. **Transform the ìWinnerî Programs.** The two ìwinnerî programs are then transformed probabilistically thru crossover (Figure 2) and mutation.
4. **Replace the ìLoserî Programs.** Replace the ìloserî programs in the population with the transformed ìwinnerî programs. The winners of the tournament remain in the population unchanged.

Iterate Until Convergence. Repeat steps two through four until a program is developed that predicts the behavior sufficiently.

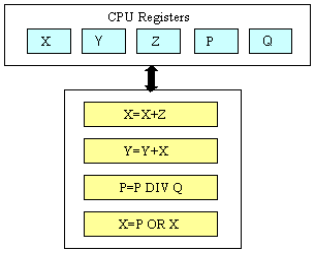

Figure 1. Genome representation in linear genetic programming

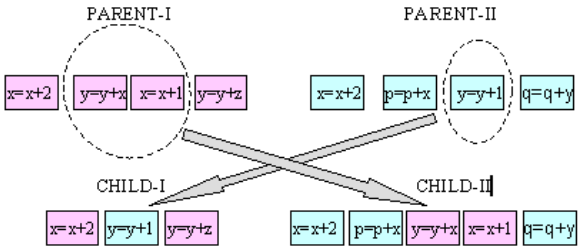

Figure 2. Crossover in linear genetic programming

One of the most serious problems of standard genetic programming is the convergence of the population. It has been often observed that unless convergence is achieved within certain number of generations, the system will never converge. Parallelization of genetic programming may hold answer to this problem by maintaining diversity in the population. Parallel populations or *demes* may possess different parameter settings that can be explored simultaneously, or they may cooperate with the same set of parameters, while each working on different individuals. However the concept is true for all EC techniques, due to their population based approach. The technique used for our LGP forecasting experiment uses circular movement of evolved programs among the demes, i.e., program movement can take place only between adjacent demes in the circle. Figure 3 depicts this concept.

Another major issue is to generate compact solutions those are more robust, i.e., generalize better on unknown data. One approach to address this can be in the form

of introducing *parsimony pressure*. This causes the natural selection process to favour shorter programs. However, researches as regards parsimony pressure are not fully conclusive. Constant parsimony pressure often tends to lead towards trapping in local optima, whereas, variable or adaptive parsimony pressure may produce better results. Further information on GP/LGP can be found in [1] and [7].

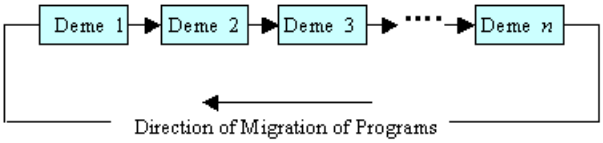

Figure 3. Migration among demes

2.2 Artificial Neural Network (ANN)

Artificial neural networks were designed to mimic the characteristics of the biological neurons in the human brain and nervous system. An artificial neural network creates a model of neurons and the connections between them, and trains it to associate output neurons with input neurons [12, 9]. The network ìlearnsî by adjusting the interconnections (called weights) between layers. When the network is adequately trained, it is able to generate relevant output for a set of input data. A valuable property of neural networks is that of generalisation, whereby a trained neural network is able to provide a correct matching in the form of output data for a set of previously unseen input data. We used the Backpropagation (BP) algorithm and the Scaled Conjugate Gradient Algorithm (SCGA) [17] for training the neural network. More details about BP and SCGA techniques could be found in [9].

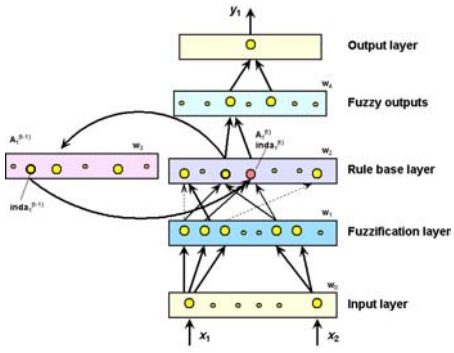

Figure 4. Architecture of EFuNN

2.3 Neuro-Fuzzy System

An integrated neuro-fuzzy system is a combination of neural network and Fuzzy Inference System (FIS) [10] in such a way that neural network learning algorithms are used to determine the parameters of FIS [11]. We used Evolving Fuzzy Neural Network (EFuNN) [12] implementing a Mamdani type FIS [19]. Referring to

Figure 4, in EFuNN, the input layer is followed by the second layer of nodes representing fuzzy quantification of each input variable space. Each input variable is represented here by a group of spatially arranged neurons to represent a fuzzy quantization of this variable. The nodes representing membership functions can be modified during learning. The third layer contains rule nodes that evolve through hybrid supervised/unsupervised learning. The rule nodes represent prototypes of input-output data associations, graphically represented as an association of hyper-spheres from the fuzzy input and fuzzy output spaces. The fourth layer of neurons represents fuzzy quantification for the output variables. The fifth layer represents the real values for the output variables. EFuNN evolving algorithm was adopted from [12].

3. Experiment Setup ñ Training and Performance Evaluation

The data for our study were the recorded half-hourly actual electricity demand for the ten months period from January to November 1995 in the State of Victoria. Figure 5 shows a typical weekly cycle of electricity demand [20]. Fluctuations in daily demand are prevalent with peaks occurring around midday. Extreme weather conditions in winter and summer months accentuate peaks in electricity demand due to the widespread use of electricity for heating and cooling. Other times, electricity demand is dominated primarily by ambient temperature, time of day, working or non-working day and the day of week.

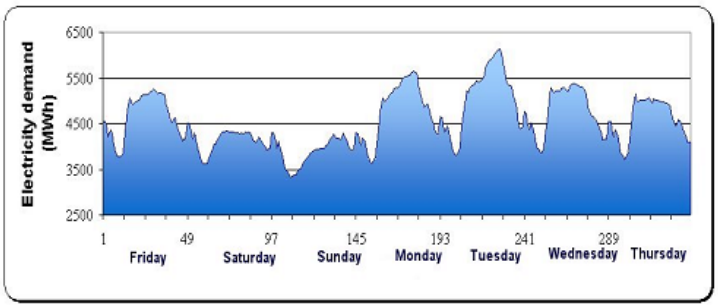

Figure 5. Typical weekly electricity demand

The experimental system consists of two stages: training the prediction systems and performance evaluation. For network training, the six selected input descriptor variables were: the *minimum* and *maximum recorded temperatures*, *previous day's demand*, a value expressing the *half-hour period of the day*, *season*, and the *day of week*. To evaluate the learning capability, the different prediction models were trained only on 20% (2937 data sets) of the randomly selected data. Our objective is to develop an efficient forecasting model capable of producing a short-term forecast of demand for electricity. The required time-resolution of the forecast is half-hourly, and the required time-span of the forecast is 2 days. This means that the system should be able to produce a forecast of electricity demand for the next

96 time periods. The training was replicated three times using three different samples of training data. We used a Pentium II, 650 MHz platform for simulating LGP based prediction models. The experiments have been repeated over different sample data sets, where only the representative results have been reported in this paper.

3.1 Genetic Programming Training

As mentioned in section 2.1, we used a linear genetic programming technique that manipulates and evolves program at the machine code level. We have used the Discipulus simulating workbench for performing the experiments using LGP [14]. The settings of various linear genetic programming system parameters are of utmost importance for successful performance of the system. This section discusses the different parameter settings used for the experiment, justification of the choices and the significances of these parameters. Table 1 gives the various parameter settings used in general in the current context. Table 2 shows the different settings used for independent experiments to identify the contributions of some of the important parameters [3] [5] [6]. The population space has been subdivided into multiple subpopulation or *demes*. Migration of individuals among the subpopulations causes evolution of the entire population. As mentioned earlier, 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.

Population size: The population size is restricted only by the random access memory (RAM) capacity of the computer. Larger the population longer will be the run-time, but may be to better effects. The effect of different population sizes (10000, 5000 and 100) has been observed.

Demes: In a population, subdivided into demes, crossover takes place both within each deme as well as between demes. Apart from the runs with a population subdivided into demes, we have also incorporated experiments with otherwise same parameter settings, but an undivided population.

Parameter	General Setting
Population Size	5000
Maximum No. of tournaments	600000
Mutation Frequency	90%
Crossover Frequency	90%
Number of demes	10
Maximum program size	256
Target subset size	100

Table 1. Parameter settings for linear genetic programming

Maximum Number of Tournaments: A genetic programming tournament denotes the process in which evolved programs are produced using the algorithm mentioned in section 2.1. The choice of the maximum number of tournaments is limited only by time and size of the RAM.

Search Parameters: 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. A constant crossover rate of 90% has been used for all the simulations. The mutation operator causes random changes in the tournament winners. We have used three different mutation frequencies viz. 90%, 45% and 10% to observe the contribution of the operator. However, many genetic programming systems use either no or significantly low mutation.

Parameter Setting No.	Population Size	Mutation Rate %	Number of Demes
1	5000	As in table 1	As in table 1
2	10000	As in table 1	As in table 1
3	100	As in table 1	As in table 1
4 or 1	As in table 1	90	As in table 1
5	As in table 1	45	As in table 1
6	As in table 1	10	As in table 1
7 or 1	As in table 1	As in table 1	10
8	As in table 1	As in table 1	No demes & no DSS

Table 2. Parameter settings for different LGP experiments

Reproduction places the copy of a program in the population, in addition to the original program. In the current context the reproduction frequency is represented by the function:

[100-(mutation-frequency)-(crossover-frequency(1-mutation-frequency))]*.

Dynamic Subset Selection: The dynamic subset selection uses a subset of the training set to help evolve more generalized solutions. The working subset is changed periodically to avoid memorization. The key factor in choosing a target subset size is that it should be representative of the original data set. Our training data set contained 2937 instances, of which only a fraction of the data has been taken as the target subset size.

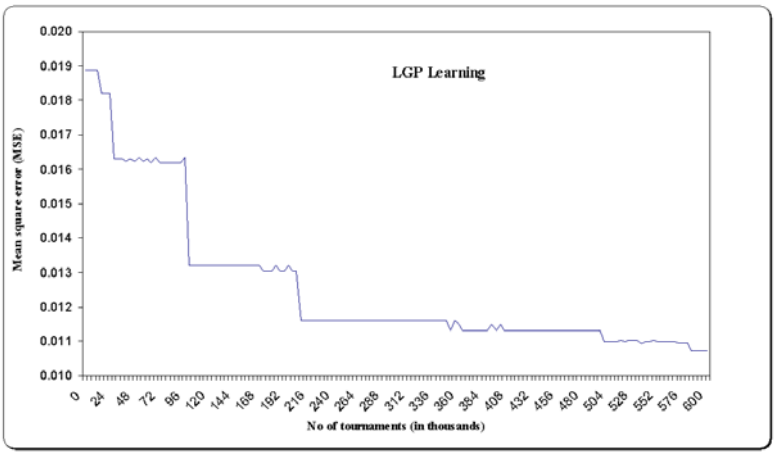

Figure 6. LGP Training results

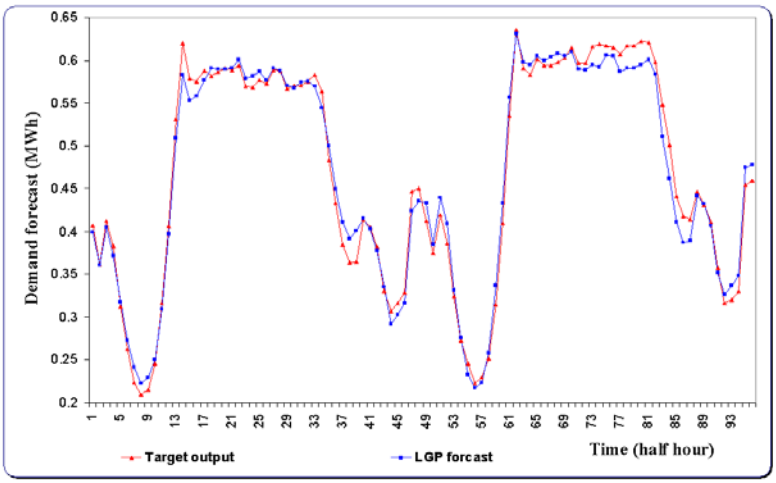

Figure 7. LGP forecasting on test data

The training performance is summarised in Table 3. Figure 6 depicts the convergence of LGP over best training fitness. Performance of linear genetic programming over test data is depicted in Figure 7.

3.2 Neuro-Fuzzy and Neural Network Training

We used 4 Gaussian membership functions for each input variable and the following evolving parameters: sensitivity threshold *Sthr* = 0.99, error threshold *Errth r*= 0.001 and learning rates for first and second layer = 0.05. EFuNN uses a one pass training approach. The network parameters were determined using a trial

and error approach. The training was repeated three times after reinitialising the network and the worst errors were reported. Online learning in EFuNN resulted in creating 2122 rule nodes. While EFuNN is capable of adapting the architecture according to the problem, we had to perform some initial experiments to decide the architecture, activation functions and learning parameters of the artificial neural network. Our preliminary experiments helped us to formulate a feedforward neural network with 1 input layer, 2 hidden layers and an output layer [6-40-40-1]. Input layer consists of 6 neurons corresponding to the input variables. The first and second hidden layers consist of 40 neurons respectively using tanh-sigmoidal activation functions. The training and test results were adapted from [2] and is summarized in Table 5.

4. Test Results: Comparisons And Analysis

Table 3 lists the comparative performance of linear genetic programming with various parameters setting as mentioned in Table 2. Table 5 summarizes the comparative performance of linear genetic programming, neuro-fuzzy system and artificial neural networks.

Parameter Setting	Training error (RMSE)	Testing error (RMSE)	Computation time(in min.)	Average program length (in bytes)
1	0.0979	0.0285	3.20	110
2	0.1359	0.0289	4	117
3	0.1004	0.0162	2.50	110
4 or 1	0.0979	0.0285	3.20	142
5	0.0974	0.02924	4	150
6	0.1308	0.0296	3.29	115
7 or 1	0.0979	0.0285	3.20	110
8	0.1035	0.0299	54	92

Table 3. Test results and performance comparison of different settings

As is observed from the test results, the linear genetic programming technique produces satisfactory results for the current forecasting problem. However, based on a single application it is illogical to make generic comments on the performance of linear GP technique and its comparability to other techniques. Furthermore, choices of parameters for this linear genetic programming based experiment were guided by the notion of getting optimal results and not necessarily for optimal runtime and also repeating the experiments with similar parameter settings does not really guarantee reproducibility of the results. This is a generic problem of evolutionary computation techniques [16]. Experimental runs over the eight test

settings (Table 2) have lead to few interesting observations regarding the performance of the linear genetic programming technique.

LGP runs with different mutation rates of 90%, 45% and 10% shows that the results improves on RMSE value with relatively higher rate of mutation. However, program length and run time shows no improvement. This could be due to the more aggressive searching caused by higher rate of mutation. Moreover exchanging a variable with mutation can have significant effect on the program flow, which perhaps is another cause of better performance with higher rate of mutation. However, the proper balance or the ratio between the mutation and crossover rate is usually a very problem dependent criterion. In this current experiment, we have simply varied the mutation rate as against a fixed crossover rate of 90%. Best performance has been obtained with a mutation rate of 45%.

We have tried the runs with different population sizes, namely 100, 5000 and 10000. Generally speaking, bigger population takes more time to evolve a generation. Furthermore, larger population-size ensures greater diversity in the population and larger search space to explore. Hence it may even reduce the number of generations needed to reach the desired solution. It has been observed by researchers that as the complexity of the problem and number of training cases increases, a larger population size is expected to produce better results. However, for certain applications, desired results can be obtained even with smaller population-size. This argument is supported by our experiment results, as they fail to show any decisive trend in performance as a result of diverse population sizes. A much smaller population size of 100 individuals have delivered the best performance in terms of RMSE value over test data.

Another important observation has been made with the use of demes. The test results for setting 1 (with and without demes), as reported in Table 4 shows clear improvement on effective training time (*the number of generations needed to reach the minimum validation error*) with minimal difference in performance. The best generation has been reached much before the specified final generation. The underlying causes can be as below. On the other hand, comparing the results obtained for Setting 1(with demes and dynamic subset selection) and Setting 8(without demes or dynamic subset selection), a very interesting trend may be observed. Without the use of demes and dynamic subset selection, the absolute computation time dramatically hikes to 54 min., as compared to 3.20 min. when they are used. However, a nominal decrease in the average program length has been observed, Figure 8.

Demes are expected to contribute in two ways:

i) Counteract the effect of local trapping in one deme by other demes with better search directions.

ii) Speed up convergence process. Perhaps the elitist migration among demes is the major contributor to this.

It has been also observed that considerable variations occur in the LGP performance over the test and training data sets. Interestingly, in all the cases, training errors have been higher as compared to the testing errors. As can be seen from Table 1, we have considered a relatively very low initial maximum program size of 30, as compared to the maximum size of 256. The results however, show that the average generated program sizes vary from 90 to 150 (Table 3). A less complex problem can benefit from lower initial maximum program size, as better individuals can be developed piece by piece.

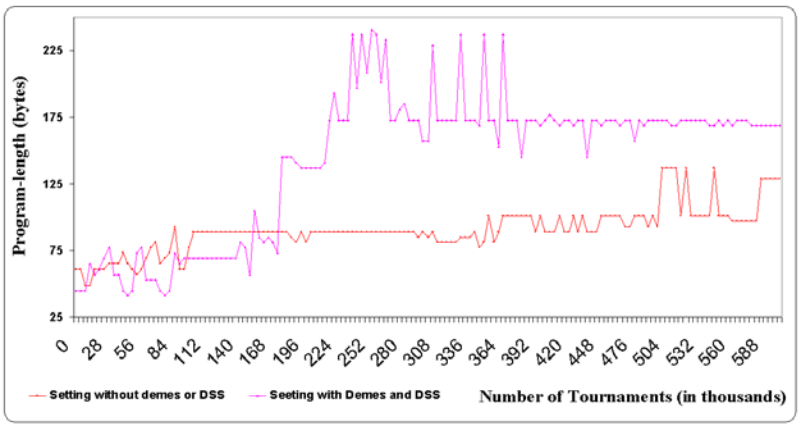

Figure 8. Growth in program length

	Setting 1 (without demes)	**Setting 1 (with demes)**
Minimum achievable training RMSE	0.0957	0.0979
Minimum achievable testing RMSE	0.0299	0.0285
Effective training time (Tournaments)	500000	450000

Table 4. Effect of demes over effective training time

As regards LGPís performance (measured in terms of training and test RMSE errors and the effective training time) as compared to the evolving neuro-fuzzy system, ANN (with BP training) and ANN (with SCGA training), the following observations have been made:

Experimentation results reveal that the neuro-fuzzy model performed better than other techniques in terms of low RMSE error and fast performance. Linear genetic programming has lower RMSE error as compared to neural networks trained using BP and SCGA. The important drawback with the conventional design of neural network is that the designer has to specify the number of neurons, their distribution

over several layers, interconnection between them, initial weights, type of learning algorithm and parameters. In contrast genetic programming conceptually requires no such designing of architecture.

Method / Parameter	LGP	Neuro-Fuzzy	ANN (BP)	ANN (SCGA)
Tournaments	600000	-	-	-
Learning epochs	-	1	2500	2500
Training error (RMSE)	0.1004	0.0013	0.116	0.034
Testing error (RMSE)	0.0162	0.0092	0.118	0.0323
*Computational load		0.536	87.2	175.0

** Measured in billion flops on a Pentium II, 450 MHz Machine with 0.256 GB of RAM*

Table 5. Test results and performance comparison of different methods

The apparent superior performance of the neuro-fuzzy system may be attributed to the following arguments. Neuro-fuzzy system makes use of the linguistic knowledge of fuzzy inference system and the learning capability of neural networks. Hence the neuro-fuzzy system is able to precisely model the uncertainty and imprecision within the data as well as to incorporate the learning ability of neural networks. Even though the performance of neuro-fuzzy systems is dependent on the problem domain, very often the results are better while compared to pure neural network approach. Compared to neural networks, an important advantage of neuro-fuzzy systems is its reasoning ability (*if-then* rules) of any particular state. A fully trained EFuNN could be replaced by a set of *if-then* rules, which could be easily interpreted [15].

Our experiments on three separate data samples reveal that the results are moderately dependent on the data sample. We used only 20% of the total data to evaluate the learning capability of the considered models. Performance could have been further improved by providing more training data. Another interesting fact about the considered models is their robustness and capability to handle noisy data that are typical in power generating systems, and therefore should be more reliable in worst situations. In this paper, we have considered a forecasting time resolution of 30 minutes. Considering the rapid fluctuation of demand and for a more effective energy management, one might have to consider even a low resolution forecasting (e.g. 5 minutes).

5. Conclusions and Future Research Direction

In this paper we tried to explore the suitability of linear genetic programming technique to generate a forecasting model for power demand in Victoria, while comparing its performance with a neural network trained using BP and SCGA and

neuro-fuzzy system implementing a Mamdani fuzzy inference system. LGP has generated considerably good results in terms of RMSE and effective learning time, while the neuro-fuzzy system produced overall better results. We have also noted some interesting performance features of the linear genetic programming technique with regard to this application. One of the constraints faced was the inconsistency in the performances of the GP runs with different mutation rates and initial populations. Though it facilitates choosing the best solution, discarding the mediocre ones, multiple runs are highly computationally expensive. As is mentioned earlier, based on a single application, it is inappropriate to draw generic conclusions on the efficiency of a system, but the fact remains that a lot remains to be done to further improve the GP technique. GP is one of the few techniques, suitable to evolve structures. However, optimisation of constants within GP is a cumbersome process. Possibilities of hybridisation of genetic programming technique, by incorporating other evolutionary techniques or even some direct search techniques for optimisation of constants, are already being explored. Hybridisation may hold the key to better performance.

References

[1] W. Banzhaf, P. Nordin, R.E. Keller, F.D. Francone, Genetic Programming ñ An Introduction ñ On The Automatic Evolution of Computer Programs and Its Applications, Morgan Kaufmann Publishers, Inc., 1998.

[2] A. Abraham, B. Nath, A Neuro-Fuzzy Approach for Forecasting Electricity Demand in Victoria, Applied Soft Computing Journal, Elsevier Science, Volume 1 /2, 2001, pp. 127-138.

[3] L.M. Deschaine, F.A. Zafran, J.J. Patel, D. Amick, R. Pettit, SAIC, F.D. Francone, P. Nordin, E. Dilkes, L.V. Fausett, Solving the Unsolved-Using Machine Learningto Model a Complex Production Process-Case Example Applying Three Machine Learning Techniques, Society for Computer Simulationís Advanced Simulation Technology Conference, Washington, DC, USA, April 2000.

[4] J.R. Koza, F.H. Bennett, D. Andre, M.A. Keane, GENETIC PROGRAMMING-III ñ Darwinian Invention and Problem Solving, Morgan Kaufmann Publishers, Inc., 1999

[5] F.D. Francone, P. Nordin, W. Banzhaf, E. DILKES, L.M. Deschaine, AIM Learning^{TM} Adaptive, Real-Time, Control Technologies. Society for Computer Simulationís Advanced Simulation Technology Conference, Washington, DC, USA, April 2000.

[6] M. Brameier, W. Banzhaf, A comparison of linear genetic programming and neural networks in medical data mining, Evolutionary Computation, IEEE Transactions on, Volume: 5(1), 2001, pp. 17 ñ26.

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

[8] The Victorian ESI Reforming the Victorian Electricity Supply Industry - 1995: Industry Review.

[9] S. Haykin, Neural Networks: A Comprehensive Foundation, Second edition, Prentice Hall Inc, USA, 1999.

[10] V. Cherkassky, Fuzzy Inference Systems*: A Critical Review*, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, Kayak O, Zadeh LA et al (Eds.), Springer, 1998, pp.177-197.

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

[12] N. Kasabov, Evolving Fuzzy Neural Networks - Algorithms, Applications and Biological Motivation, in Yamakawa T and Matsumoto G (Eds), Methodologies for the Conception, Design and Application of Soft Computing, World Scientific, 1998, pp. 271-274.

[13] National Electricity Market Management Company Ltd: http://www.nemmco.com.au/

[14] AIMLearning Technology, http://www.aimlearning.com.

[15] Kasabov, N. and Woodford B. Rule Insertion and Rule Extraction from Evolving Fuzzy Neural Networks: Algorithms and Applications for Building Adaptive, Intelligent Expert Systems, I*n* Proceedings of the FUZZ-IEEE'99 International Conference on Fuzzy Systems*,* Seoul, Korea, 1999, pp. 1406-1411.

[16] P. Nordin, A Compiling Genetic Programming System that Directly Manipulates the Machine Code*,* Advances in Genetic Programming, K. Kinnear (eds.), MIT Press, Cambridge, MA, 1994, pp.311-331.

[17] A. F. Moller, A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning, Neural Networks, Volume (6), 1993, pp. 525-533.

[18] Zadeh LA, Roles of Soft Computing and Fuzzy Logic in the Conception, Design and Deployment of Information/Intelligent Systems, Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications, O Kaynak, LA Zadeh, B Turksen, IJ Rudas (Eds.), 1998, pp 1-9.

[19] Mamdani E H and Assilian S, An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller, International Journal of Man-Machine Studies, Vol. 7 (1), 1975, pp. 1-13.

[20] B. Nath and M. Nath, Using Neural Networks and Statistical Methods for Forecasting Electricity Demand in Victoria, International Journal of Management and Systems, Volume 16 (1), 2000, pp. 105-112.