Benchmarking a Recurrent Linear GP Model on Prediction and Control Problems

Xiao Luo, Malcolm Heywood, and A. Nur Zincir-Heywood

Faculty of Computer Science, Dalhousie University
Halifax, NS, B3H1W5, Canada

Abstract. In this work, a recurrent linear GP model is designed by introducing the concept of internal state to the standard linear Genetic Programming (GP), so that it has the capacity of working on temporal sequence data. We benchmarked this model over four standard prediction and control problems, which include generic even parity problem, sun spot series prediction, Lorenz Chaotic time series prediction and pole balance control problem. From the experimental results, the recurrent linear GP model appears to be very competitive compared to those algorithms relying on spatial reasoning of the temporal problem.

1 Introduction

Genetic Programming has been applied to a wide range of supervised learning problems, chiefly formulated as either classification or function approximation problems [4]. They have also seen widespread application within the context of reactive environments with delayed payoff (reinforcement learning), such as the 'Ants' [4] or robot control problems. In both cases, the models relay on a spatial description of the problem. However, in this work, we are interested in problems, which have temporal descriptions. Thus, pattern sequences and capacity of detecting or retaining the temporal relationships between the patterns of a sequence is now important. There are two basic solutions to build models for solving temporal problems. In the first case, the temporal dependence is encoded by the features of each pattern using some *apriori* information, thus reducing the problem to spatial reasoning alone. Examples of this might involve encoding the temporal property of the problem using a sliding window (shift register) of some predefined depth and resolution. Such an approach has seen wide spread application to predictive problems [9], [10] and [11]. In the second case, a recurrent learning model is employed. This means that the model has capacity to retain state across more than one pattern. Examples of recurrent models include Hidden Markov Models and recurrent neural network models. In these examples, support for reasoning about temporal aspects of the problem is provided by feedback paths internal to the model. Various evolutionary approaches have been proposed for building such models [1] and [5].

The motivation of this work is to design and implement a recurrent linearly structured GP (L-GP) model that falls in the second case and benchmark it on

predictive and control problems. In the following, the recurrent L-GP model is described in section 2. Results from an experimental study are presented on four different problems in section 3. Finally, conclusions are drawn and future work is discussed in section 4.

2 Recurrent Linear Genetic Programming

2.1 Linear Genetic Programming

Linearly Structured Genetic Programming is based on a representation closely related to that employed by Genetic Algorithms. Specifically, individuals are constructed from a (linear) sequence of integers each of which has to be decoded into a valid instruction (syntactic closure). The decoding process effectively translates each integer into instruction. In this work, a 2-address instruction format is employed e.g., $R1 = R1 + IP3$, where R1 denotes a register with index '1' and $IP3$ is a reference to a input with index '3' or 3rd feature of the current input pattern. The specific form of linearly structured GP(L-GP) employed by this work utilizes the page-based L-GP developed in an earlier work [3]. Such a scheme enforces a fixed length representation, the basic components of which are defined as follows.

- *Representation*: Individuals take the form of a 2-address instruction format. Individuals are described in terms of a (uniform) randomly selected number of pages, where each page has the same fixed number of instructions.
- *Initialization*: Individuals are described in terms of the number of pages and instructions. The number of pages per individual is determined through uniform selection over the interval [1 ... maxPages]. Defining an instruction is a two-stage process in which the mode bit is first defined(instruction type) using a roulette wheel(user specifies the proportions of the three instruction types). Secondly the content of the remaining fields is completed with uniform probability.
- *Selection Operators*: A steady-state tournament is employed. In this case all such tournaments are conducted with 4 individuals selected from the population with uniform probability. The two fittest individuals are retained and reproduce. The children over-write the worst two individuals from the same tournament using their respective position in the population.
- *Variation Operators*: Three variation operators are utilized, each with a corresponding probability of application. Crossover selects one page from each offspring and swaps them. Mutation has two forms. The first case is referred to as 'Mutation'which merely Ex-OR's an instruction with a new instruction. The second mutation operator is denoted as 'Swap' which identifies two instructions with uniform probability in the same individual and interchanges them.

This represents the basic page-based L-GP scheme. However, the selection of the page size is problem specific. As a consequence the Dynamic Page based L-GP algorithm was introduced to modify the number of instructions per page

dynamically during the course of the training cycle [3]. Such a scheme was demonstrated to be much more robust than that of a fixed page size over a range of benchmark problems [3]. In this work, the Dynamic Page based L-GP algorithm is employed.

2.2 Recurrent L-GP

The only modification necessary to change a standard L-GP model into a recurrent model is to retain register values between sequential pattern presentations. Thus within the context of a prediction problem, the registers are never reset until the last pattern of the input sequence is reached. The prediction read from the predefined 'output' register(s) is to predict the next pattern following the last pattern that input into L-GP. In the case of a control problem in which a failure state might be explicitly reached, the model would be allowed to run until such a state occurs and then the registers are reset before a new initial condition is selected and the process is repeated.

3 Experiments and Results

A total of four benchmark problems are considered from a recurrent modeling context: Generic solution to the even parity problem; predictor for the Lorenz chaotic attractor; predictor for the sun spot series problem, and a controller for the pole balance problem. The GP learning parameters are summarized in Table 1.

Table 1. GP Learning Parameters

Data set	Parity, Sun Spot, Pole Balance	Lorenz
Pop. Size	125	
Max. Instr.	128	512
Max Tournaments	50000	500000
Num. Reg.	4	8
Function Set	+, -, *, %	
Terminal Set	{0, . . . , 255}$\bigcup${input index}	
P(Xover)	0.9	
P(Mutation)	0.5	
P(Swap)	0.9	
Runs	50(25 on second pole balance problem)	

3.1 Generic Even Parity

The even parity problem is a well-known early benchmark in which the basic objective was to derive a specific even parity instance using 2 input logical op-

erators that excluded Ex-OR [4]. Here, our objective is to derive 6- and 7-parity from a training set consisting of 2-, 3-, 4- and 5-parity. The input sequence consists of bits associated with the parity case. The bits are input into recurrent L-GP one after another. On presentation of the last bit in the sequence the value of register R0 which acted as "output" register is compared to the label for that sequence. Fitness function is the sum square error of all training sequences.

The simplest(and most typical) solution generated by recurrent L-GP to this problem consisted of only two instructions: $R0(t) = R0(t-1) - X(t)$; $R0(t) = R(t) \times R(t)$. X is the input sequence and $X(t = 1 \ldots n)$ corresponds the input bit in the sequence. R0 is initialized to 0 ($R0(0) = 0$). It is worth to mention that not only this solution is a very concise solution for 6- and 7-parity sequences but also for all n-parity sequences.

3.2 Sun Spot Time Series

The Sun Spot time series has been a benchmark prediction problem in a number of studies. The typical approach has been to use a sliding window with length n to go through the whole sequence to construct a spatial presentation of the sequence, then a predictive model f is built to predict the next time step $n+1$, i.e. $x(t+n+1) = f(x(t), \ldots, x(t+n))$. In our system, no 'n' is predefined, all the patterns before $x(t+1)$ are input to predict the $x(t+1)$. Thus, t is dynamic. This leaves the selection of relevant previous time steps to the recurrent L-GP model. In line with previous work, the dataset is divided into training(221 patterns representing the years 1700-1920), and two test sets (Test set 1 has 35 patterns (1921-1955), Test set 2 has 24 patterns (1956-1979)). Fitness function takes the form of a normalized mean square error as shown in formula 1.

$$NMSE(P) = \frac{1}{\sigma^2}\frac{1}{P}\sum_{p=1}^{P}(desired(p) - GPout(p))^2 \tag{1}$$

Where $\sigma^2 = 1535$ and P is the pattern count for the dataset [9]. The best solution for this problem consists of 35 instructions. Table 2 provides a comparison of recurrent L-GP with other predictors identified in previous works on the same dataset based on their best performance. The other models provide lower errors on training and the first test partition, but degrade significantly on the second test partition, which represents the period most distant from the training partition. It is important to notice that recurrent L-GP is more consistent over the training and 2 test sets than the other approaches.

3.3 Lorenz Chaotic Attractor

Prediction of a chaotic time series has also been widely used benchmark for predictive models. Same as the case of the Sun Spot benchmark, the typical approach for this problem is to build a predictor based on the sliding window [7]. Lorenz Chaotic time series is defined over three variables by the discrete differential system,

Table 2. Comparative Results on Sun Spot Problem

Model	**NMSE (train)**	**NMSE (test1)**	**NMSE (test2)**
Recurrent L-GP	0.1077	0.1655	0.1708
NN [11]	0.082	0.086	0.35
TAR [9]	0.097	0.097	0.28
Recurrent NN [5]	0.1006	0.0972	0.4361
GP [10]	0.125 ± 0.006	0.182 ± 0.037	0.370 ± 0.06

$$\dot{x} = \sigma(y - x); \dot{y} = \rho x - y - yz; \dot{z} = xy - bz \tag{2}$$

where $\sigma = 10, \rho = 28$,and $b = 8/3$.

The time series is built from an initial condition of (0, 0.01, 0) and a step size of 0.01. A total of 4000 samples from the sequence are constructed with the first 2000 discarded to avoid any start up properties. The remaining 2000 samples are then divided equally between training and test. The fitness function is the normalized mean square error as shown in formula 1. The best results for this problem are NMSE(training)= 1.38×10^{-5} and NMSE(test)= 1.09×10^{-5}. The best solution for this problem consists of 48 instructions. In comparison, the SOM based Dynamic Learning architecture of Principe et al., produced training errors of the order 0.0011 [7], obviously worse than the performance of the recurrent L-GP model.

3.4 Pole Balance

The pole balance or inverted pendulum problem places the learning system within the role of a bang-bang controller [6]. The controller supplies a control force of $\pm 10N$ to a cart on which an inverted pendulum is connected. Cart behavior is described in terms of x - the distance from the center of a track on which the cart travels -, and θ - the angle of the pendulum relative to the vertical. The state of the cart can be described in the form of a binary fail (unbalanced)/no fail (balanced) metric, where the failure state is defined by the condition,

IF ($|\theta|$ >12) OR ($|x|$ >2.4) THEN (fail) ELSE (no fail)

The objective here is to evolve recurrent L-GP to produce a controller to supply force to the cart so as to keep the cart balanced as long as possible. Given a force produced by a controller, the behavior of the cart is described by a series of differential equations, modeled as an Euler discrete event simulation at a step size of 0.01 [6].

We set up two sets of experiments. One has two 2 inputs (x and θ), the other has 4 inputs (x, θ and their corresponding velocities). In line with previous GP solutions to this problem, for the training process, we set the 10 initial states of the cart (x, θ and their corresponding velocities) randomly over the interval ± 0.2. We did 25 different random seed runs, all of them converged. The pole could last 8 seconds under all 10 random initializations without failure in training for both conditions of 2 and 4 inputs. In comparison with the controller evolved by Chellapilla using a macro-mutation

operator based Tree structured variant of GP, the mean time a pole was balanced during training was 2.7228 seconds, with no controller solving the problem [2]. Hence, our system is significantly better.

4 Conclusions

In this paper, we describe a recurrent linearly structured GP model and benchmark it on a series of prediction and control problems. This model works on sequence directly instead of using a sliding window to map the temporal data representation into spatial data representation. The results show that this model is more consistent on both the training and the test sets than the sliding window based model. Moreover, the evolved rule generated by the recurrent linear GP is a set of instructions, which are very simple. Future work will investigate the inter-relationships between solution lengths and the size of the register set and new fitness functions, which are able to express the temporal nature of the problem. We are also interested in applying such a model to text sequence analysis, and bio-oriented sequence analysis.

References

1. Angeline, P.J., Saunders, G.M., Pollack, J.B.: An Evolutionary Algorithm that Constructs Recurrent Neural Networks, IEEE Transactions on Neural Networks. 5(1), (1994) 54–64.
2. Chellapilla, K., Evolving Compuer Programs Without Subtree Crossover. IEEE Transactions on Evolutionary Computation. 1(3), (1997) 209–216.
3. Heywood, M.I., Zincir-Heywood, A.N.: Dynamic Page-Based Linear Genetic Programming, IEEE Transactions on Systems, Man and Cybernetics - PartB: Cybernetics. 32(3), (2002)380–388.
4. Koza, J.R.: Genetic Programming: ON the Programming of Computers by Means of Natural Selection, MIT Press, MA, (1992)
5. McDonnell, J.R., Waagen, D.: Evolving recurrent Perceptrons for Time-Series Modeling, IEEE Transactions on Neural Networks. 5(1), (1994) 24–38.
6. Miller, W.T., Sutton, R.S., Werbos, P.J.: Neural Networks for Control. MIT Press,(1990)
7. Oakley H.: Two Scientific Applications of Genetic Programming: Stack Filters and Non-Linear equation Fitting to Chaotic Data. In Advances in Genetic Programming. Chapter 17. K.E. Kinnear (ed). MIT Press, MA, (1994) 369–390.
8. Teller, A.: The Evolution of Mental Models. In Advances in Genetic Programming. Chapter 9. K.E. Kinnear (ed). MIT Press, MA, (1994)198–219.
9. Tong, H., Lin, K.S.: Threshold autoregression, limit cycles and cyclical data. J. of the Royal Statistical Society. B 42,(1980) 245.
10. Vallejo, E.E., Ramos R.: Evolving Turing Machines for Biosequence Recognition and Analysis. In Proceedings of the 4th European Conference on Genetic Programming (EuroGP 01). Springer Verlag, Berlin,(2001) 192–203.
11. Weigend, A.S., Huberman, B.A., Rumelhart, R.E.: Predicting the Future: A Connectionist Approach. Int. J. of Neural Systems. 1(3), (1990) 193–209.