Gene Expression Programming For Induction of Finite Transducer

Jandhyala Seetha Manognya
School of Electrical & Electronic Engineering,
Nanyang Technological University,
Singapore
smanognya@gmail.com

A/P Lipo Wang
School of Electrical & Electronic Engineering,
Nanyang Technological University,
Singapore
ELPWang@ntu.edu.sg

Abstract — **This paper presents an alternative method for solving the problem of finite transducers using gene expression programming (GEP). Each individual in the GEP system represents a Mealy machine with outputs for each state. Be means of roulette-wheel sampling, individuals are chosen for the next generation and are put through a series of genetic operators which seek to change the mark-up of the individual to better fit the selection environment/ fitness sets. The system was tested with five problems to show its effectiveness and success at solving all of those problems.**

1 INTRODUCTION

A finite transducer or sequential machine is a state machine which is used to model the behaviour of a system. It contains a finite number of states, state transitions and actions associated with each transition. It is a model of a machine with primitive internal memory. Each of its state transition is labelled with two symbols, one for input and the other for output. Although the finite transducer is widely used in image processing and natural language processing, there has been limited research done in using GEP to evolve effective state solutions. This paper presents a new way to inducing finite transducers by using gene expression programming.

This paper is organized as follows: Section 2 presents a general idea of finite transducers. Section 3 gives an overview on gene expression programming. It also briefly addresses previous algorithms which attempted to evolve finite state machines. Section 4 proposes the GEP system for Mealy machines. Section 5 presents the test problems used. The results attained are discussed in Section 6. Finally, section 7 concludes with the findings of the study.

2 FINITE TRANSDUCERS

There are two types of finite transducers- Mealy machines and Moore machines. In a Mealy machine, the output depends on both the external inputs and the present state, whereas in a Moore machine, the output depends only on the present state of the system.

Formally defined, Mealy machines can be represented by the equation below:

$$Me = \{q0, \Sigma, \Gamma, Q, \delta\}$$

where
- $q0$ is the start state
- Σ is the input string alphabet
- Γ is the output string alphabet
- Q is the set of states
- δ: $Q * \Sigma * \Gamma \rightarrow Q$, i.e. defines the transitions [1]

The operation of a Mealy machine can be conveniently described by a state diagram, which diagrammatically defines the State-to-State transition of a sequential machine.

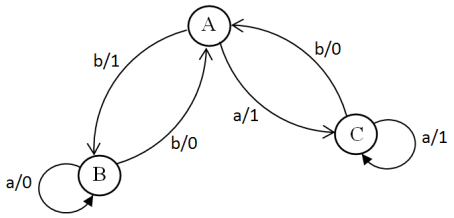


Figure 1. State Diagram of Mealy Machine. Starting at state A, if the input to the system is a, the system would move to state C to perform some operation, giving an output of 1. If the input at A was b, the system would transact to state B, with an output of 1.

Algorithms exist to convert Mealy machines to Moore machines [1]. Hence, only the induction of Mealy machines would be examined here in this paper.

3 GENE EXPRESSION PROGRAMMING

Gene Expression Programming, proposed by Ferreira, is an evolutionary algorithm that mimics biological evolution to solve a user defined problem. Each generation contains a population of individuals or chromosomes which are linear

978-1-4244-4657-5/09/$25.00 ©2009 IEEE

structures with fixed lengths (genotype). The solution can be decoded by translating them into an expression trees (ET) (phenotype). With explicit head and tail distinctions, the chromosomes are easier to be modified genetically, posing no ambiguity and coding no invalid structures. Each of the head portion is made up of functions and terminals, i.e. when solving the problem of symbolic regression, the functions can be any of the mathematical operator. A terminal is the argument of the function. The tail portion consists of only terminals.

The ET can be derived by looking at the chromosome itself. Hence, the ET does not take part in the evolution process. What is passed on to the next generation is only the linear genetic mark-up of the chromosomes and not the tree structure.

The basic genetic operators used in GEP are selection, replication, mutation, inversion, transposition and recombination.

In order for evolution to take place, the full range of operators are applied to generation n and is measured for fitness against the selection environment or fitness cases, which are the inputs to the system. The best of generation n are selected by means of roulette-wheel sampling for replication to generation n+1.

Dupont [2] utilized genetic algorithm to encourage Deterministic Finite Automata's (DFA) to evolve. His methodology involved taking the largest, most basic automaton of the group of positive sentences which make up the input of the language [1]. Dunay et al. [3] and Brave [4] used different genetic programming (GP) approaches to evolve automata encodings of the DFAs. While the previously mentioned authors conducted their study in finite automata, Lucas [5] and Naidoo and Pillay [1] exploited the area of finite transducers using GP as well. The former method uses a table corresponding to a transition matrix and the later a transition graph of states to find the potential solution.

Although GP effectively generates the state machines, GEP is proven to be faster. In addition, in GEP, all the modifications on the chromosomes always code for valid structures, whereas in GP this is not always the case for all the chromosomes. Fig.2 shows a study by Zhou, Xiao, Nelson and Tirpak comparing the average expression size and the execution time generated by GEP and GP over ten different runs on five benchmark data sets. [6]

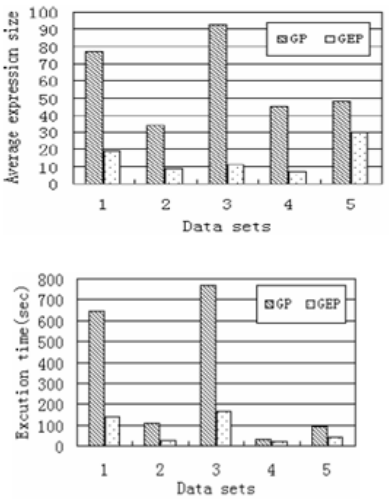


Figure 2. Comparison between the average expression size and execution time between GP and GEP

GP uses the ETs as its chromosome. This means that GP works at the tree level where each chromosome is a ramified structure of different size and shape. There is no distinction between the head and the tail portion of the chromosome and often, structurally invalid chromosomes are formed during modifications, making it unusable. Hence, not surprisingly, huge initial population of parse trees are needed to feed them with the necessary building blocks in order to evolve good solutions by shuffling the blocks around [9]. This problem is overcome in GEP with the implementation of distinctive head and tail portions in the chromosomes. The tail containing exclusively only terminals can also be looked at as a reserve of terminals, ensuring that the gene always codes for valid structures. Hence, regardless of any kind of modification, every chromosome is acceptable. This implicates that with GEP, effective and good solutions can be reached even with a small population.

Another point to emphasize is that in GP, the whole chromosome has to be the solution to the problem. There is no, say, a cut-off region to limit the answer as and when needed. This implies that, a high level of human assistance is needed to make sure the predicted values of the parameters are ideal. Over extrapolation of the values may not allow the answer to converge to a definite solution. In the GEP, on the other hand, there is a coding and non-coding region in the chromosome which would determine until which portion is the answer valid, ignoring the rest. This reduces the load on the accuracy of inputs to the system as they can be ignored when in the non-coding regions. For instance, if a state machine only has 3 states and the input for GP and GEP is 4, the 4th state in the tree of a GP chromosome makes the solution invalid; yet, its presence in the non-coding region of a GEP chromosome would not affect the solution in any manner. This gives GEP another edge over GP in terms of a wider range of value selection for the parameters.

It can thus be concluded that GEP is definitely more superior, in terms of computational capability and complex

function handling, and faster at arriving at the solution. The following section presents the methodology of using GEP for Mealy machines.

4 THE PROPOSED GEP SYSTEM

The basic chromosome for the GEP system is represented in a linear structure of alphabets, for states and numbers, for outputs as shown below.

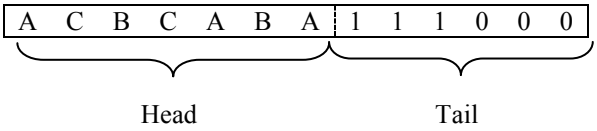


Figure 3. Structure of Gene

Evolution, by means of the genetic operators, is carried out on the chromosomes.

Mutation operates on single elements, changing the elements to other elements, whereas inversion works on a series of elements, reversing the order of the elements in the chromosome. The transposition operator functions to copy a portion of the chromosome and paste it in another location. The same number of copied elements is also deleted at the end of the head/tail to maintain the structural composition of the chromosome. The difference in insertion sequence (IS) and Root IS transposition lies in the position where the copied elements are placed. Recombination is a process where the elements of two chromosomes crossover at one or two randomly chosen points to form two child chromosomes. Except for mutation, the rest of the operators are performed on selected individuals in the population, determined by the rates of reproduction.

Modifications we propose to the GEP system invented by Ferreira [9] include:

1. The linking function is removed from the system as it is not possible to link two or more sub-state systems using one fixed state to form the ultimate state machine. Since a state needs both inputs and outputs to be defined, they could not be gotten from the sub-ETs.
2. This also means that the multi-gene chromosomes are not possible.
3. With a single gene, gene transposition and gene recombination operators are redundant.
4. The functions are the predicted number of states and the terminals are the output values. Represented as an ET, each state is connected to another state, not an output. Hence, there can only be functions in the head portion of the gene and only terminals in the tail. The head length, number of states and the number of variables for each state are linked by the equation:

   *head_length = (no_of_states * no_of_indp_var) + 1*

   For example, for the state diagram in fig.1, the 3 states and 2 inputs for each state leads to a head length of 7.
5. The tail length would therefore add up to be head_length-1.
6. For the transposition operators, the transposition is restricted to the head and the tail independently. Reason being that the elements of head and tail cannot be accidentally transposed into the tail and head portions respectively. This means that when a chromosome is randomly chosen to undergo transposition, it is also randomly decided whether the operation would be performed on the head or the tail.
7. The fitness cases are not comprised of, say, 10 independent sample data. Since a continuum of inputs and states are required to determine the output of the state machine, the fitness cases are also in continuous sets of inputs and outputs. The maximum fitness is achieved when all the sets are matched by the GEP generated state machine. The table below shows an example of fitness sets with each set comprising of 3 inputs and outputs, corresponding to the machine in fig.1.

TABLE 1. SAMPLE FITNESS CASES

Set	Input	Output
1	0	1
	0	1
	0	1
2	0	1
	0	1
	1	0

8. The fitness function is a simple comparison of the target value and the predicted value of output from the state machine. If he predicted value is the same as the target value, then the chromosome is fit for that particular input /output combination, else, it isn't (fitness=0).
9. In an ET, the left-most branch is the first variable input and so on. The ET for the state machine in fig.1 is depicted below.

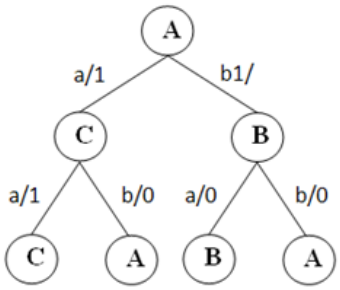


Figure 4. Expression Tree

10. There is one more round of translation of the results from the ET's to the state diagrams.

5 TEST PROBLEMS

This paper presents a different approach to solving the problem of finite transducers using GEP. Hence, the proposed GEP is tested on some of the standard finite transducer benchmarks described in the general literature on theory of machines and formal languages such as [7] [1]. The benchmarks are listed in Table 2.

TABLE 2. MEALY MACHINE DATA SET [7] [8]

Machine	Description	Example
M1	Mealy machine that outputs a 1 for every substring 'aaa'	Input:bbaaababba Output: 0000100000
M2	Mealy machine that outputs a 1 for every substring 'aab'	Input:baaabaabba Output: 0000100100
M3	Mealy machine that outputs a 1 for every double letter	Input:baaabaabba Output: 0011001010
M4	Mealy machine that takes in a binary input and outputs its 1's complement	Input: 0011001010 Output: 1100110101
M5	Mealy machine that takes a binary input and outputs E's and O's in places where the number of 1's read in so far are even and odd, respectively	Input: 1100110101 Output: OEEEOEEOOE

Matlab version 7.0 was used to implement the system and run on Windows based 1.86 GHz Laptop with 512 MB of RAM.

The table below shows the GEP parameters used during the tests. Note that the rates of genetic operators are chosen based on prediction and as suggested by Ferreira [9]. Also, those parameters indicated by an asterisk (*) are changed with every test. The number of fitness sets for each test range from 16 to 30. The termination point of the system is when all the sets match with the generated Mealy machine or when the maximum number of generations is reached.

TABLE 3. INPUT PARAMETERS TO GEP SYSTEM

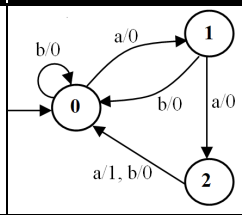
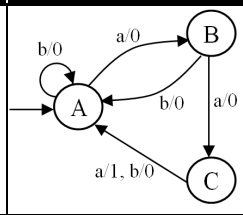
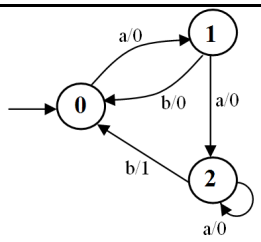
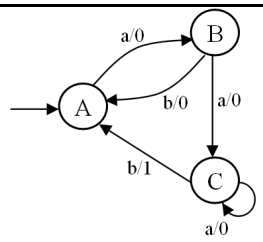
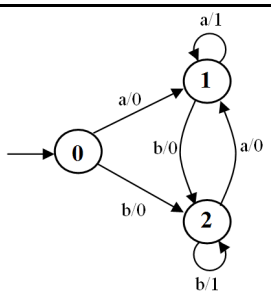
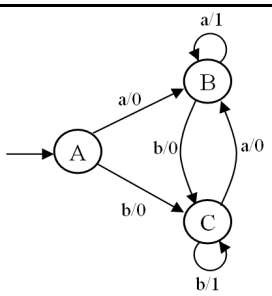
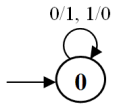
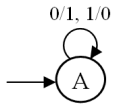
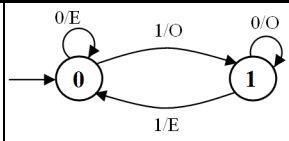
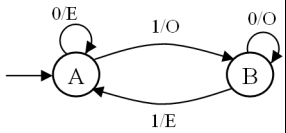
Input Parameters	
No. of states *	3
No. of variables per state *	2
No. of fitness cases per set *	5
Population Size	20
No. of generations	100
Mutation Rate	0.2
Inversion Rate	0.1
IS transposition Rate	0.1
RIS transposition Rate	0.1
One-point recombination Rate	0.4
Two-point recombination Rate	0.2

6 RESULTS AND DISCUSSION

The proposed GEP was applied to the data in Table 2 to generate five machines, each of which perfectly matches the requirements of each machine. Each test solution was evolved within 50 generations, although that cannot be said for more complicated systems which might have more than 3 states and those that need more input sets to generate the correct machine. The machines were also human generated to see if the results match. They were compared with those presented by Naidoo and Pillay [1] using genetic programming. It must be taken into account that although the results are compatible with those obtained using GP, with the use of GEP, the solution is attained much faster. The study results in fig.2 help to reiterate this point that with a shorter execution time and a smaller expression size, GEP is a better choice of algorithm for finite transducer problems.

Table 4 shows the result presented by Naidoo and Pillay and those generated by GEP. It is worth noting that all of the machines are exactly the same with both methods. This also proves the reliability of GEP.

TABLE 4. RESULTS GENERATED BY GP AND GEP

	GP Generated Solution	GEP Generated Solution
M1		
M2		
M3		
M4		
M5		

7 CONCLUSION

The objective of the research presented in this paper is to see whether GEP could be used to evolve induction finite transducers. The proposed GEP was constructed and tested on five standard transducer problems. They results were not only compared with the human generated machines but also with the GP generated machines and proved to be exactly the same. This reflects on GEP being a very promising algorithm to be used in the areas of computer science. Although both GP and GEP can be used to arrive at the same result, the outcome can be reached much faster with the use of GEP and it also has the capacity to compute more complex problems. Further study is required to fully maximize the capabilities of GEP with the help of multi-gene chromosomes.

REFERENCES

[1] Pillay, A. N. (2007). The Induction of Finite Transducers Using Genetic Programming. In Lecture Notes in Computer Science (pp. 371-380). Valencia, Spain: Springer.

[2] Dupont, P.: Regular Grammatical Inference from Positive and Negative Samples by
Genetic Search: the GIG Method. In Carrasco, R.C. and Oncina, J. (eds.): Grammatical
Inference and Applications (ICGI-94). Springer-Berlin, Heidelberg (1994) 236 - 245.

[3] Dunay, B.D.: Petry, F.E., Buckles, B.P: Regular Language Induction with Genetic
Programming. In: Proceedings of the 1994 IEEE World Congress on Computational
Intelligence, Orlando, Florida, USA. IEEE Press (1994) 396 - 400.

[4] Brave, S.: Evolving Deterministic Finite Automata Using Cellular Encoding. In : J.R. Koza
et al. (eds.): Proceedings of the First Annual Conference on Genetic Programming (GP 96).
MIT Press (1996) 39 - 44.

[5] Lucas, S. M.: Evolving Finite State Transducers: Some Initial Explorations. In: Genetic
Programming: 6th European Conference, EuroGP 2003, Essex, UK, April 14 -16, 2003,
Lecture Notes in Computer Science, Vol. 2610. Springer (2003) 130 - 141.

[6] Chi Zhou, Weimin Xiao, Peter C. Nelson, and Thomas M. Tirpak: Evolving Accurate and Compact Classification Rules with Gene Expression Programming. IEEE Transactions on Evolutionary Computation, Vol. 7, No. 6 (2003) 519–531

[7] Cohen, D. I. A.: Introduction to Computer Theory, John Wiley & Sons (1986).

[8] Forcada, M.L.: Neural Networks: Automata and Formal Methods of Computation, January
http://www.dlsi.ua.es/~mlf/nnafmc/pbook.pdf. (2002).

[9] Candida Ferreira (May 2006). Gene Expression Programming: Mathematical Modelling by an Artificial Intelligence. Springer. 77-1