TREAD: A New Genetic Programming Representation Aimed at Research of Long Term Complexity Growth

Tony E Lewis
School of Computer Science and Information Systems
Birkbeck, University of London
London, UK
tony@dcs.bbk.ac.uk

George D Magoulas
School of Computer Science and Information Systems
Birkbeck, University of London
London, UK
gmagoulas@dcs.bbk.ac.uk

ABSTRACT

Several forms of computer program (or representation) have been proposed for Genetic Programming (GP) systems to evolve, such as linear, tree based or graph based. Typically, GP representations are highly effective during the initial search phases of evolution but stagnate before deep levels of complexity are acquired. A new representation, TREAD, is proposed to combine aspects of flow of execution and flow of data systems. The distinguishing features of TREAD are designed for researching improvements to the long term acquisition of novel features in GP (at the expense of the speed of the initial search if necessary). TREAD is validated on a symbolic regression problem and is found to be capable of successfully developing solutions through artificial evolution.

Categories and Subject Descriptors

I.2.2 [**Artificial Intelligence**]: Automatic Programming—*program synthesis*

General Terms

Algorithms, Design, Experimentation

Keywords

Artificial Intelligence, Genetic Programming, Representations

1. INTRODUCTION

The prospect of solving complex problems by artificially evolving highly complex computer programs for many generations appears more attainable recently with the advances in CPU and GPU technologies. An obstacle, however, appears to be that although GP systems tend to be powerful in the initial search stages they tend to stagnate before developing significant complexity [1].

This work is concerned with the role of representation on a GP system's ability to continue acquiring useful features and proposes a new representation which is designed to facilitate research of these issues.

The term "representation" is used in this paper in a strict fashion to refer only to the architecture and its interpretation. Techniques such as indexed memory (which is used to achieve Turing completeness in the PADO system [3]), grammar guiding, and strong typing are not included under the term.

Representations in the GP literature are often classified into linear, tree and graph representations [1]. A common feature of most representation designs is that they allow the language (the set of functions and terminals used in the representation) to be varied according to the problem domain.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

2. A NEW REPRESENTATION: TREAD

Tangle Representing Execution And Data (TREAD) is a new representation which is designed for research into tackling highly complex problems. The architecture of TREAD is a directed, possibly cyclic graph. In a strict sense, the graph represents the flow of data but it also determines the flow of execution. Its similarity to standard methods means that it is amenable to mutation and modularity using similar mechanisms.

The key priorities in the design of the representation are that it should be powerful, associated with a smooth fitness landscape and programmable and extendable by humans. It is hard to imagine manually adding successive layers of useful complexity to a tree. It seems reasonable to wonder if this task is similarly difficult for artificial evolution. It is the intention that a TREAD program is easier to continue developing.

2.1 TREAD's Features

A TREAD program consists of a potentially cyclic graph of nodes which is evaluated in an iterated fashion. This makes it similar to Neural Programming, CGP [2] (with cycles permitted) and PDGP (with cycles permitted). TREAD's key additions, which bear some similarity to aspects of an artificial neural network (ANN), are:

- **A state of execution.** In contrast to other architectures, TREAD's graph represents both the flow of data and the flow of execution. This is achieved by each (non-terminal) node having a state of execution which can be either of two values: *executing*, meaning the node continues to compute updated outputs for each iteration or *not executing* meaning the node simply repeats the previous output. The state of execution is determined by an extra input to the node - the *trigger socket*. A non-negative input to the trigger socket causes the node to enter the executing state and a negative value causes it to enter the not executing state. Any type of output socket may be connected to

any type of input socket and each node has an output *activated socket* indicating its state of execution. The activated socket outputs a value of one if the node is executing or zero otherwise. The input trigger socket can be conveniently depicted on the top of a node and the activated socket on the bottom. A terminal node does not have a trigger socket and its activated socket always outputs a value of one.

On the one hand, TREAD aims to exploit the power of flow of execution representations; on the other hand it aims for a smoother fitness landscape through the redundancy of having many parts of the graph executing simultaneously rather than having a single path of execution. To further encourage a powerful representation with a smooth fitness landscape, the mutation operator is allowed to add and remove function nodes.

- **Multiple connections per input socket.** In the TREAD architecture, each input socket (including the trigger socket) is allowed more than one input connection. The value used by the input socket is the sum of the values coming from each input connection. This means that the graph structure is particularly unrestricted as any socket (input or output) can have many connections (and any output socket may be connected to any input socket). The aim of this feature is to encourage useful additions to developing TREAD programs by allowing new parts of the graph to be added to the input of a node.

- **Weighted connections.** The TREAD architecture allows for each connection to have an associated (possibly negative) weight. The mutation operator may make quantitative changes to a TREAD program via the weights of the connections (in addition to qualitative changes to the structure of the program). The value supplied by a connection to the relevant input socket is calculated as the value it receives from the relevant output socket multiplied by the connection's weight. Hence if a particular input socket has n connections with weights $w_1, w_2, \ldots, w_n$, which are receiving output values $o_1, o_2, \ldots, o_n$ respectively, then the value at the input socket will be calculated as $\sum_{i=1}^{n} w_i o_i$. This feature encourages a smooth fitness landscape: the intuition is that a new part of a TREAD graph with a low weight contribution might have enough evolutionary feedback to improve without completely destroying the reproduction potential of the whole program and that the contribution from a part can be increased when it has become advantageous.

3. EXPERIMENTAL VALIDATION

The aim of this preliminary experiment is to establish that TREAD is amenable to improvement through artificial evolution and that it can begin to tackle a typical GP problem. To this end, a GP system has been set up to evolve TREAD programs to tackle a symbolic regression problem using the same formula as was used to first test CGP [2]: $x^6 - 2x^4 + x^2$. The terminal set is $\{x, 1\}$ and the function set is $\{+, -, *, \%\}$ (where % represents protected division which operates like a normal division except that the result is 0 if the denominator is 0). The fitness cases are the 50 uniformly distributed numbers between -1.0 and 1.0 inclusive. The fitness of an individual is calculated as the negative of the sum of the absolute differences between the prediction and the correct answer given by the formula. The negation in this fitness definition is so that higher fitness values are better.

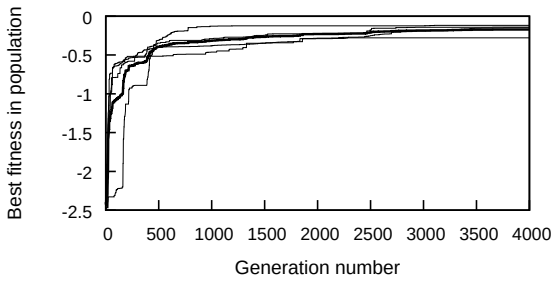

Figure 1: Best fitness evolution for the symbolic regression problem. Thin lines denote different runs, while the bold line represents the average over those runs.

As can be seen from Figure 1, the experiment demonstrates that TREAD is indeed amenable to improvement through artificial evolution and able to begin to tackle a typical GP problem.

4. CONCLUSIONS AND FUTURE WORK

TREAD is a new representation that combines information about the program data and the program execution in an attempt to construct a representation for research into long term complexity growth in GP. It has been demonstrated that TREAD can be evolved successfully (albeit slowly) to tackle symbolic regression - a typical GP problem.

It is hoped that future research will allow TREAD to make new progress into problems that other representations find difficult. However, the version of TREAD presented here is expected to be less efficient at finding correct solutions to problems that other representations can solve. For example, CGP has already proved itself to be an effective representation at solving problems such as this instance of symbolic regression.

It is hoped that it will be possible to accelerate TREAD's evaluations through the use of GPU computation.

5. REFERENCES

[1] W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA, Jan. 1998.

[2] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[3] A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, 1996.