

Solving Symbolic Regression Problems Using Incremental Evaluation In Genetic Programming

¹ School of ITEE, University of New South Wales
@ Australian Defence Force Academy, Canberra, Australia.
Email: Hao: t.hao@adfa.edu.au

² School of Computer Science & Engineering, College of Engineering, Seoul National University, San 56-1, Sinlim-dong, Gwanak-gu, Seoul 151-744, Korea

³ Vietnamese Military Technical Academy, Hanoi, Vietnam.
Email: nxhoai@gmail.com

Abstract—In this paper, we show some experimental results using Incremental Evaluation with Tree Adjoining Grammar Guided Genetic Programming (DEVTAG) on two symbolic regression problems, a benchmark polynomial fitting problem in genetic programming, and a Fourier series problem (sawtooth problem). In our pilot study, we compare results with standard Genetic Programming (GP) and the original Tree Adjoining Grammar Guided Genetic Programming. Our results on the two problems are good, outperforming both standard GP and the original TAG3P.

I. INTRODUCTION

Genetic Programming (GP) was investigated by Koza in 1992 [2]. It is an automatic learning methodology using simulation of evolution to discover functional programs to solve a problem. Genetic programming breeds a population of trial solutions using biologically inspired operators, which include reproduction, crossover (sexual recombination) and mutation, combined with selection. In essence, it uses evolutionary search methods to search an in-principle unbounded space of expressions for solutions to given problems. However the solutions found are generally poorly structured and highly disorganised, exhibiting no hierarchical or modular structure. An individual in a genetic programming system is generally expected to solve problems immediately, without benefit of a developmental phase. In contrast, the natural evolutionary systems on which it is based are able to evolve hierarchical modular structure (e.g. the homoeobox gene complex). Generating hierarchical, modular structures would greatly benefit GP, potentially dramatically increasing the scalability of GP

application, as well as the adaptability of GP solutions. In fact, many approaches have been investigated for developmental GP; for example, Angeline and Pollack developed an alternative technique called Module Acquisition [3] which is based on the creation and administration of a library of modules for the automatic generation of subroutines. Other studies have investigated Automatically Defined Functions (ADF) – based GP [4], which is probably the most popular of the modularization methods currently used in GP. In ADFs, along with a main tree, additional branches are maintained from the root of the tree which define ADFs. An ADF is called just as if it belonged to the primitive set. Thus the hierarchy of components is fixed; despite this, it boosts the power of solution discovery, especially for problems with regular solutions, or those decomposable into smaller sub-problems.

Recently, interest in developmental approaches in Evolvable Hardware has begun to increase. Haddow et al. used Lindenmayer systems for digital circuit design [6], while Miller developed Cartesian Genetic Programming for the automatic evolution of digital circuits, and attempted to evolve a cell that could construct a larger program by iteration of the cell's program [7].

Nevertheless, hierarchical structure has not been clearly demonstrated in existing developmental GP systems. We argue that this is because modular structure, if used for a single evaluation as in most artificial developmental systems, only has adaptive advantages to entire species, not to particular individuals, and hence cannot be selected for by evolution. In developmental biological systems, on the other hand, evaluation is continuous throughout development (if the individual is insufficiently fit to survive at a particular stage of development, the fitness it would exhibit at later stages is immaterial). A modular structure, which allows biological sub-systems to develop in synchrony throughout development, can thus provide a selective advantage to the individual. Our working hypothesis is that, if the individual is evaluated on multiple problems at different stages of development, then modular structure can provide an

Tuan Hao Hoang is with the University of New South Wales @ ADFA, Canberra ACT 2600 Australia (tel +61 2 62688693, fax +61 2 6268 8581, email: t.hao@adfa.edu.au)

Daryl Essam is with the University of New South Wales @ ADFA, Canberra ACT 2600 Australia. email: d.essam@adfa.edu.au

Bob McKay is with Seoul National University, Seoul 151744, Korea; email: rim@cse.snu.ac.kr

Xuan Hoai Nguyen is with the Vietnamese Military Technical Academy, Hanoi, Vietnam, email: nxhoai@gmail.com

[This is a self-archived copy of the accepted paper, self-archived under IEEE policy. The authoritative, published version can be found at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1688570&tag=1](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1688570&tag=1)

adaptive advantage to that particular individual, and hence can be selected for by evolution. This hypothesis is investigated using the representation of the Tree Adjoining Grammar Guided GP system (TAG3P) which has useful properties for supporting evaluation during incremental development. In particular, TAG has a feasibility property, allowing any TAG expression tree to be evaluated regardless of the detachment of any number of its sub-trees. This means that smaller sections of the tree can easily be tested on simpler problems, providing a straightforward way to test our hypothesis at relatively low implementation cost.

Preliminary results on the polynomial Symbolic Regression problem using incremental evaluation in TAG [11] have been interesting, and encouraged us to do further experiments on that problem, and also on a more difficult problem, the sawtooth wave – Fourier series problem.

The paper outline is as follows. The next section briefly describes Tree Adjoining Grammars (TAGs), and TAG based Genetic Programming. Section 3 introduces our Incremental Evaluation method based on Tree Adjoining Grammar Guided Genetic Programming (DEVTAG) and reviews our earlier work on the Symbolic Regression problem. Experimental setups are described in section 4. Section 5 and 6 provide the results and discussion. Conclusion and future work are laid out in the last section.

II. TREE ADJOINING GRAMMAR- TREE BASED GENETIC PROGRAMMING

A. Tree Adjoining Grammar

Tree adjoining grammars (TAGs) are tree-generating and analysis systems, first proposed by Joshi et al in [8]. Tree Adjoining Grammars (TAGs) have become increasingly important in Natural Language Processing (NLP) since their introduction.

The aim of TAGs is to more directly represent the structure of natural languages than is possible in Chomsky languages, and in particular, to represent the process by which natural language sentences can be built up from a relatively small set of basic linguistic units by inclusion of insertable sub-structures. Thus ‘The cat sat on the mat’ becomes ‘The big black cat sat lazily on the comfortable mat which it had commandeered’ by the subsequent insertion of the elements ‘big’, ‘black’, ‘lazily’, ‘comfortable’, and ‘which it had commandeered’. This process is much more directly represented in TAGs than in the better-known Context Free Grammar (CFGs).

In more detail, a tree-adjoining grammar comprises a quintuple (T, V, I, A, S), where:

- T is a finite set of terminal symbols.
- V is a finite set of non-terminal symbols ($T \sqcup V = \Sigma$).
- S \in V is a distinguished symbol called the start symbol.

- I are initial trees, which are characterized by: all interior nodes being labeled by non-terminal symbols, while the nodes on the frontier are labeled either by terminal or non-terminal symbols. Non-terminal symbols on the frontier of an initial tree are marked with a down arrow (\downarrow).

- A are auxiliary trees, which are characterized by all internal nodes being labeled by non-terminal symbols, while nodes on the frontier are labeled either by terminal or non-terminal symbols. Amongst the non-terminal-labeled symbols on the frontier, there is one special one, the foot node. The foot node must be labeled with the same non-terminal symbol as that labeling the tree’s root node. The convention of marking the foot node with an asterisk (*) is followed here.

The trees in $E = I \sqcup A$ are called elementary trees. Initial trees and auxiliary trees are indicated as \diamond and \heartsuit respectively; and a node labeled by a non-terminal (terminal) symbol is usually called a non-terminal (terminal) node. A tree with root labeled by non-terminal symbol X is called an X-type elementary tree.

Lexicalized TAGs (LTAGs) contain at least one terminal node in each of their elementary trees. It has been proven that for any context-free grammar G, there exists an LTAG G_{lex} that generates the same language and tree set as G. The derivation trees in G are the derived trees of G_{lex} .

The key operation used with tree-adjoining grammars is the adjunction of trees. A brief description follows; more details can be seen in [1].

Adjunction builds a new (derived) tree \heartsuit from an auxiliary tree \heartsuit and a tree \diamond (which may be initial, auxiliary or derived) by inserting \heartsuit into \diamond at an appropriate place. More formally, if a tree \diamond has an interior node labeled A, and \heartsuit is an A-type tree, the adjunction of \heartsuit into \diamond to produce \heartsuit is as follows: Firstly, the sub-tree \diamond_1 rooted at A is temporarily disconnected from \diamond . Next, \heartsuit is attached to \diamond to replace the sub-tree. Finally, \diamond_1 is attached back to the foot node of \heartsuit . \heartsuit is the final derived tree achieved from this process. Adjunction is illustrated in Figure 1.

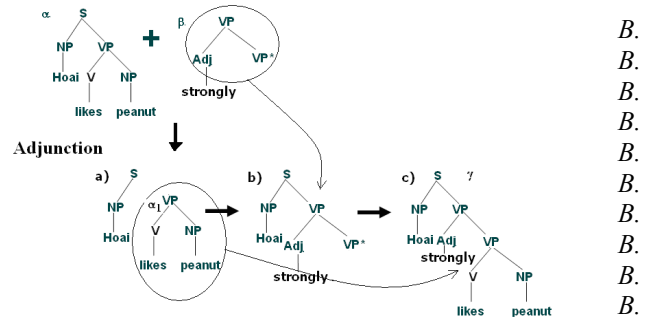


Fig. 1. An Example of Adjunction Operator

TAG based Genetic Programming

Tree Ajoining Grammar Guided Genetic Programming (TAG3P) [1] is a grammar guided genetic programming system using context-free grammars and tree-adjoining

grammars to set syntactical constraints and search bias for the evolution of programs. An important property of TAG representation is a feasibility property, thanks to which, in growing a derivation tree from the root, one can stop at any time and still have a valid derivation tree, as well as a valid derived tree. Feasibility helps TAG3P to control the exact size of its chromosomes, as well as to implement a wide range of new operators, both biologically inspired – including relocation [1] and duplication [12] – and also operators such as point insertion and deletion, motivated mainly by their local search effects.

The TAG3P system from [1] has five main components:

1) *Parameters*: The minimum and maximum sizes of genomes (MIN_SIZE, MAX_SIZE), size of population (POP_SIZE), maximum number of generations (MAX_GEN) and probabilities for genetic operators.

2) *Initialisation procedure*: The process of randomly generating an LTAG derivation tree begins with choosing a random size between MINSIZE and MAXSIZE, and then selecting a random \diamond -tree from the initial tree set to make an initial G_{lex} derivation tree. This derivation tree is subsequently augmented by \diamond -trees randomly drawn from the auxiliary tree set in G_{lex} by using adjunction at random places. This process repeats until the chosen size is reached.

3) *Fitness evaluation*: Each individual in TAG3P is translated into a derived context-free grammar tree first. After that, the evaluation is computed on that derived tree in the same way as in Context-Free Grammar-Guided Genetic Programming: the function defined by the leaf nodes is evaluated. The search space is thus defined by the grammar - the set of all GP expression trees which may be generated by the given grammar, within the specified complexity bound. However, unlike most other tree-based GP systems, because of the feasibility property, tree size rather than depth is used as the complexity bound.

4) *Selection methodology and reproduction*: In TAG3P, a selection methodology is performed before applying any other genetic operators. To do this, TAG3P selects one (mutation) or two (crossover) individual programs from the population with a probability based on their fitness measures. The selection operator used in this paper is tournament selection. In the tournament selection method, a fixed number of programs or trees are chosen at random, and the one with the smallest fitness value wins.

5) *Crossover*: The crossover process occurs between two parent individuals, namely two G_{lex} derivation trees $t1$ and $t2$ chosen from the population by applying the selection method. First, two nodes in the two trees may be accepted if the sub-tree under each can adjoin to the parent node of the other. Random sampling of nodes in each tree is repeated, until either two such nodes are successfully found in $t1$ and $t2$, or the number of trial exceeds a predefined bound. In the second case, crossover will not happen. After that, if two such nodes are found, the crossover is completed by swapping the sub-trees beneath these two nodes (there is some additional book-keeping, as in most GP systems, to

ensure that the individuals produced by crossover satisfy the condition that the size of the individuals produced lie between MINSIZE and MAXSIZE)

6) *Mutation*: For the mutation operation, a sub-tree of a tree is chosen at random. It is then removed and replaced by a new sub-tree of the same size generated in the same manner as the initialization procedure.

III. INCREMENTAL EVALUATION BASED ON TAG3P

This paper continues our previous pilot study [11] of a preliminary developmental evaluation system based on TAG3P. In this work, the developmental process is minimal, as our focus at this stage is still on the evaluation aspect of developmental evaluation. The developmental process consists simply of revealing more of the genotype of the individual at each stage of development – conceptually, this corresponds to the simple developmental processes of lower organisms, without the feedback loops and complexities of higher organisms.

To take a concrete example, one of the problem classes considered in this paper arises from fitting Fourier series to a sawtooth wave (or from a physical perspective, from taking the outputs of a series of low-pass filters of increasing frequency). A sawtooth wave may be approximated by finite segments of the Fourier series $Y = \sin X + (1/2) \sin(2 \cdot X) + (1/3) \sin(3 \cdot X) + \dots$. Suppose we are given randomly sampled (X, Y) values from some of the lower terms in this series. Can we, without relying on the information that this is a Fourier series, recover the underlying functions? In effect, we are attempting to solve a family of symbolic regression problems.

We define

$$F1 = \sin X$$

$$F2 = \sin X + (1/2) \sin(2 \cdot X)$$

$$F3 = \sin X + (1/2) \sin(2 \cdot X) + (1/3) \sin(3 \cdot X)$$

$$F4 = \sin X + (1/2) \sin(2 \cdot X) + (1/3) \sin(3 \cdot X) + (1/4) \sin(4 \cdot X)$$

...

$$F9 = \sin X + (1/2) \sin(2 \cdot X) + (1/3) \sin(3 \cdot X) + (1/4) \sin(4 \cdot X) + (1/5) \sin(5 \cdot X) + (1/6) \sin(6 \cdot X) + (1/7) \sin(7 \cdot X) + (1/8) \sin(8 \cdot X) + (1/9) \sin(9 \cdot X)$$

We aim to exploit the gradual and systematic increase in problem difficulty to solve the final (quite difficult) symbolic regression problem. Koza [2] and others have studied the discovery of Fourier series by GP; our aim is very different, using Fourier series as a way of defining a family of related problems. The much weaker biases of our approach render the results difficult to compare fairly.

To achieve this, the individual is separated into layers of fixed depth (corresponding to the stages of the developmental process). For the simplest problems, only shallow depths of the individual are used (corresponding to young biological organisms coping with limited environmental challenges). Increasingly more of the individual is used to handle more complex problems (corresponding to an individual handling more challenging environments as it grows and ages). The ability to do this is

a consequence of the feasibility property of the TAG3P representation, that any rooted subtree of a valid TAG3P tree is also a valid TAG3P tree. Thus if we truncate the tree to a given depth, we still have a tree which can be evaluated. In the particular problem family considered here, the tree might be divided as follows:

- Depth 2 used to solve function F1
- Depth 7 used to solve function F2
- Depth 12 used to solve function F3
- ...
- Depth 42 used to solve function F9

(Note that the schedule of depths is one of the necessary parameters of this approach; we consider this issue in more depth in the discussion).

We use tournament selection, which only requires a fitness partial ordering of individuals. For DEVTAG, we use a special multi-stage comparison to generate this ordering. Corresponding to the biological insight that later-stage fitness is only important if the individual survives earlier stages, we compare individuals on simpler problems first; only if they are roughly equivalent on the simpler problems do we evaluate them on more complex ones.

We denote the fitness of an individual I evaluated at stage j by $F(I,j)$. For two individuals (I_1, I_2) , the comparison process (for minimisation) is:

```

i := 1;
While |F(I1, i) - F(I2, i)| < ε
  i := i + 1;
if (F(I1, i) < F(I2, i))
  then I1 wins
  else I2 wins

```

An example of this algorithm is shown in Figure 2, comparing the individuals I_1 and I_2 with fitness value arrays (corresponding to the 9 different stages), $I_1(10.05, 14.67\dots, 20.35)$, and $I_2(10.06, 14.66, \dots, 10.35)$. In this case, I_2 would be chosen for evolution.

IV. EXPERIMENTAL SETUPS

As in most grammar-based GP systems, the search space is delineated by a grammar. The context-free grammar G for the first experiments in this paper has a function set including unary and binary operators $\{+, -, *, /, \sin, \cos, \log, \exp\}$. The terminal sets are X and 1.0 .

Formally:
 $G = (N, T, P, S)$
 $S = \text{EXP}$ – the start symbol
 $N = \{\text{EXP}, \text{PRE}, \text{OP}, \text{VAR}, \text{CONST}\}$
 $T = \{x, 1.0, \sin, \cos, \lg, \exp, +, -, *, /\}$,
 (ep is exponential, lg is log function).

Fig. 2. An Example of comparing two individuals in DEVTAG

P consists of
 EXP \overline{X} EXP OP EXP | PRE EXP | VAR | CONST
 OP \overline{X} + | - | * | /
 PRE \overline{X} sin | cos | lg | ep
 VAR \overline{X} x
 CONST \overline{X} 1.0

G and G_{lex} are described in more detail in [11].

The first set of problems we will consider is symbolic regression for a simpler (and well-studied) family of polynomial functions, originally due to Koza [2].

- F1 = X
- F2 = X²+ X
- F3 = X³+X²+X
- F4 = X⁴+ X³+X²+X
- ...
- F9 = X⁹+X⁸+X⁷+X⁶+X⁵+X⁴+X³+X²+X

Some preliminary results using our incremental evaluation

TABLE I
 PARAMETER SETTING FOR POLYNOMIAL, (FOURIER) PROBLEMS

Objective	Find a function that fits a given sample of 20 (40) (xi, yi) data points.
Success Predicate	Sum of errors over 20 (40) points $< \sqrt{\epsilon} = 0.01$ (0.1)
Terminal sets	X – the independent variable (ONE - the constant)
Operators(Function set)	+, -, *, /, sin, cos, exp, log
Fitness Cases	The sample of 20 (40) points in the interval [-1..1] ($10 \cdot 20 \cdot \overline{X}$)).
Fitness	Sum of the errors over 20 (40) fitness cases.
Genetic Operators	Tournament selection(3), sub-tree crossover and sub-tree mutations for all systems considered
Parameters	The crossover probability is 0.9. The mutation probability is 0.1
Min/Max initial size for TAG3P, DEVTAG	2 to 1000
Max depth for GP	30
Depth schedule	2, 4, ..., 18 (2, 7, ..., 42)
Population size	250 (250)

method on this incremental problem family were presented in [11]. There, we reported that, with the same number of function evaluations, DEVTAG significantly outperformed both standard tree-based GP and the original, GP-like TAG3P. For example, in an experimental setting with

population size 250, and a budget of 229,500 function evaluations DEVTAG's probability of success was 33%, well above that achieved by the other treatments – TAG3P's probability of success was 8%, while no successes were achieved in 100 GP runs. However those experiments left open some further questions.

First, the good performance of DEVTAG could result from some other aspect of the DEVTAG system than incremental evaluation. Three further treatments were implemented, using variants of TAG3P and GP.

1) *gGP*: This treatment was designed to address a potential issue, that differences exhibited by DEVTAG might arise simply from the increasing difficulty of fitness functions in DEVTAG, independent of the developmental process. In this treatment, F1 is used for the first $229,500/9 = 25,500$ evaluations, then F2, and so on; more formally, for i from 0 to 8, generation $(i * \text{MAXGEN})$ to generation $(i+1) * \text{MAXGEN} - 1$ use fitness function $F(i+1)$. Otherwise, the treatment is identical to GP.

2) *gTAG*: The corresponding treatment to *gGP*, but applied to the TAG representation.


3) *DEVTAGF9ALL*: This treatment is designed to address the converse issue. Differences noted in DEVTAG performance might arise simply from the incremental evaluation giving an opportunity to find small solutions, without any need for changing fitness functions. This treatment uses the multi-stage evaluations of DEVTAG, but each stage is evaluated using the fitness function F9, instead of varying through the family F1 to F9.

Second, the good performance of DEVTAG could be restricted to this particular problem set, with its very highly structured solutions. To evaluate this, we chose a slightly less regular problem set (a Fourier series) as a second set of test functions. The Fourier family is also relatively regular, but this is hardly surprising. Our developmental evaluation hypothesis (that developmental evaluation is important in developing modular solutions to decomposable problems) relies on the assumption of decomposability and regularity; we do not expect the developmental evaluation approach to provide any benefit unless there is some modular structure available to be discovered.

For the Fourier problem, we chose to compare the basic three algorithms (DEVTAG, TAG and GP), the results of the previous experiments giving us no reason to continue with *gGP*, *gTAG* or *DEVTAGF9ALL*. To give comparability on node evaluations, we first ran DEVTAG for 229,500 function evaluations, to determine how many node evaluations it used. We then provided a budget of at least that number of node evaluations to the other algorithms.

A further complication arose: GP and TAG individuals frequently evaluated to NaN or INF values. Note that a tree only has to evaluate to NaN or INF on one of the 40 input instances to cause problems. Since these values cause difficulty in evaluating performance averages, we handled it by resetting them to a very bad fitness value (100).

This problem turned out to be extremely hard for all three

algorithms, though the results are favourable to DEVTAG. To simplify the problems somewhat, we ran a further three treatments denoted with the suffix '+'. These used exactly the same experimental settings as the previous three treatments, except that partial solutions in the form of eight extra  trees were added to the TAG and DEVTAG grammar, the beta trees encoding the expressions:

$$\text{beta1} = +(1/2) * \sin(2 * X)$$

$$\text{beta2} = +(1/3) * \sin(3 * X)$$

...

$$\text{beta8} = +(1/9) * \sin(9 * X)$$

In the GP treatment, we considered each of these formulae as a term with arity 0, and added it to the GP term set.

V. RESULTS

A. The polynomial symbolic regression set:

Firstly, we consider the new, more detailed, polynomial symbolic regression results. Table 2 shows the absolute number of successful runs out of 100, for each of the six treatments (the original TAG, GP and DEVTAG treatments, together with the new treatments testing alternative hypotheses for the performance of DEVTAG), using a population size of 250. Note that the 0 entries mean we have not found any successful runs using the *GP*, *gGP* or *DEVTAGF9ALL* treatments.

From these results, we can safely conclude that the performance of DEVTAG may be partly due to the representation (TAG vs GP), and may also be assisted by the increasing difficulty of fitness function (*gTAG*), but is not simply due to the Incremental process (*DEVTAGF9ALL*). It appears that the full developmental evaluation method is required to achieve the performance of DEVTAG. This is covered in more detail in the discussion section; for now, we simply note that the experiments confirm that there is little point in continuing with these treatments, they do not explain the performance of DEVTAG.

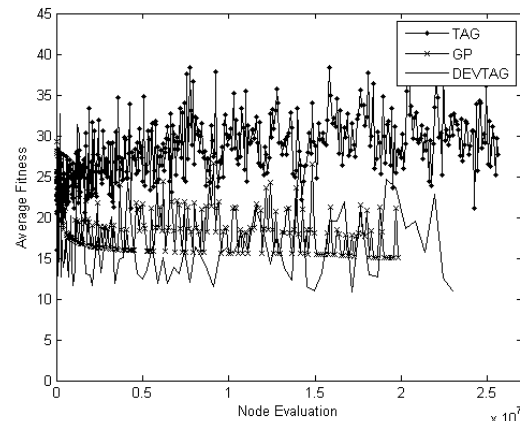


Fig. 3. Average fitness of the best individuals for Fourier problem at each generation of DEVTAG, TAG, and GP vs Total number of node evaluation

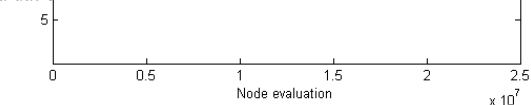


Fig. 4. Average Fitness of the best individuals for Fourier problem at each generation of 9 functions on DEVTAG vs Total number of node evaluation

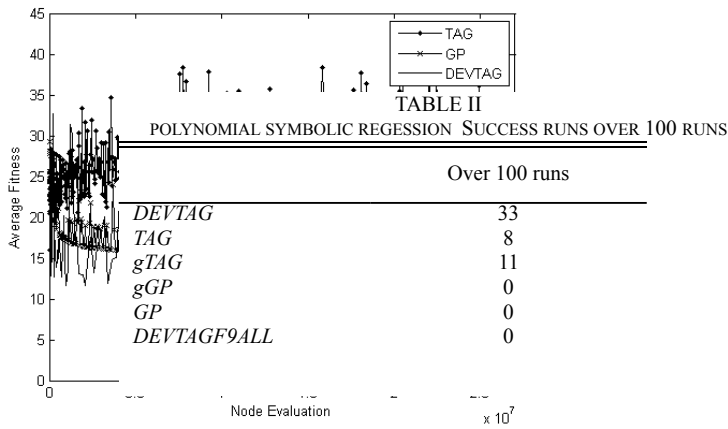


Fig. 3. Average fitness of the best individuals at each generation of FR_DEVTAG, FR_TAG, and FR_GP vs Total number of node evaluation

B. Fourier series problem:

For the Fourier series problem, Figure 3 shows the average fitness for each of the three treatments. We use a slightly non-standard presentation in these plots. In most evolutionary computation papers, the X axis represents the number of function evaluations. This is natural in the general field, where most methods have fixed evaluation cost. Even in GP, where evaluation costs may vary, it is generally reasonable because the average evaluation cost is similar between the methods being compared. However one of the primary aims of our incremental evaluation method is to evolve highly structured, compact genomes. Thus we expect the evaluation cost of our individuals to be less than that for most GP systems. To ensure a fair comparison, we compute the total number of node evaluations, so as to more accurately assess the true computational cost, and use it as the X axis in the relevant figures.

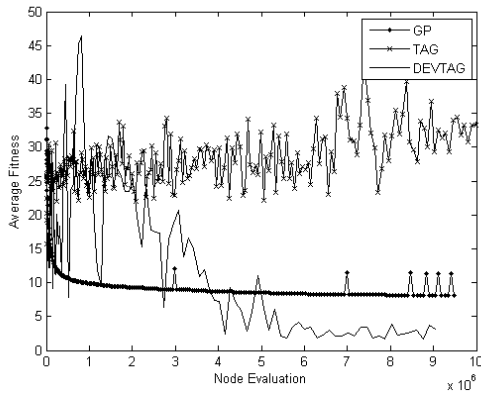


Fig. 5. Average Fitness of the best individuals at each generation of DEVTAG+, TAG+, and GP+ vs Total number of node evaluation

Each point in the plot describes the mean, over 30 runs for those treatments, of the best individual in each population at the given total number of node evaluation. Figure 4 shows the average fitness error, of some Fourier functions: F1, F2, F7, F8, and F9, of the best individual at each generation of the DEVTAG experiment. Figures 5 and 6 are parallel to Figures 3 and 4 respectively but using the additional grammar productions / terms described at the end of the previous section.

VI. DISCUSSION

From table 2, it is clear that incremental evaluation is very effective at finding exact or near-exact solutions to the Symbolic Regression problem. At population size 250, DEVTAG's probability of success was 33%, well above that achieved by any other treatment. It is also clear that this is an extremely difficult task for standard GP. It's worth noting that DEVTAG gives us solutions to all the other eight functions, at no extra computation cost.

In passing, we note also that the performance of TAG is surprisingly good, so that the representation is clearly an

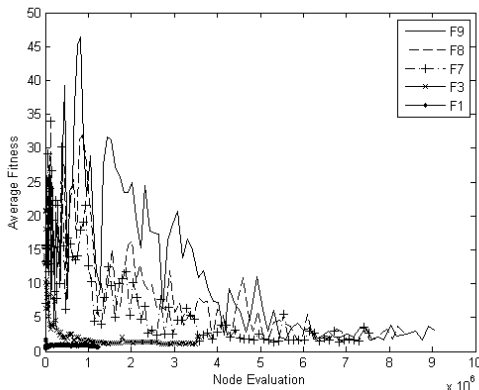


Fig. 6. Average Fitness of the best individuals at each generation of 9 functions on DEVTAG+ vs Total number of node evaluation

important contributing factor. For the polynomial symbolic regression, the *gTAG* results are slightly better than those of *TAG*, but not enough to be important. Thus, merely changing the fitness function during the process of evolution isn't the cause of the good performance of DEVTAG. On the other hand, *DEVTAGF9ALL* performs extremely poorly; we interpret this as showing that it is difficult to solve F9 in a small tree depth, and difficult to extend an approximate solution at that level to a good solution at a deeper level. That is, the results of DEVTAG are not simply the result of having an incremental process; evaluation during the incremental process is required to get good results.

From Figure 3, it is clear that the Fourier problem is extremely difficult. None of the systems (*TAG*, *GP* or *DEVTAG*) is able to solve F9, or even to get particularly close to it. It's worth noting that a trivial solution ($\sin X$) has fitness of just under 20 (depending on the 40 fitness cases) for F9; that is, the expected error of $\sin X$ on a random fitness case is just under 0.5 (the expected error of $\sin X$ on the exact sawtooth wave being exactly 0.5). In fact, the performance of *TAG* on this problem is worse than this – that is, *TAG* produces so many NaNs and INFs that it is unable even to achieve the performance of the simple function $\sin X$. On the other hand, *GP* and *DEVTAG* are able to do significantly better. While the performance curves of *GP* and *DEVTAG* are still badly affected by NaNs and INFS, the underlying fitness curves appear to asymptote to values around 16 and 12 respectively (0.4 and 0.3 expected error on each fitness case). We may certainly conclude that *GP* and *DEVTAG* out-perform *TAG* on this problem, and tentatively conclude the *DEVTAG*'s performance is better than *GP*'s. But overall, it's clear that this is an extremely difficult problem, too difficult for all of the systems to achieve good performance.

We note that an important part of this difficulty may arise from the need to find two relatively complex components in synchrony. That is, each new term of the Fourier approximation has the form $(1/n) * \sin(n*X)$. The two 'n' values, constructed from the terminals by reasonably complex trees, need to be somewhat synchronized to make a fitness contribution. This synchronization of learning may be extremely difficult. To investigate the issue, we decided to simplify the problem, avoiding synchronization requirements, leading to the final set of experiments.

On this (much easier) problem, *TAG* was still unable to make headway, average fitnesses remaining worse than those of the trivial solution $\sin X$. However *GP* and *DEVTAG* performed considerably better. Both appeared able to avoid the NaN/INF problem, with the *GP* runs asymptoting to an apparent limit fitness of around 7.5 (0.175 per fitness case), and *DEVTAG* to around 3 (0.075 error per fitness case). From Table III, we see that *DEVTAG*, in this greatly simplified problem, is able to learn F9 with a 20% success rate, while *TAG* and *GP* had no success.

As before, we note that with *DEVTAG*, we also get, with no extra computational cost, a 26.66% probability of finding

F8, 30% of finding F7, and higher probabilities of success for the simpler functions.

VII. CONCLUSIONS AND FUTURE WORKS

The results of incremental evaluation using *TAG* representation clearly demonstrate a form of problem-driven incremental evolution. *DEVTAG* has been provided with three families of related problems of increasing difficulty, and it has proceeded to solve them incrementally – fairly successfully in the two easier cases, though with only limited success in the harder case.

It does this, though, at a cost. To run our current version of *DEVTAG*, it is necessary to specify two extra parameters beyond those normally required for a *GP* system, namely the initial incremental evaluation depth, and the increment from level to level. Naturally, there is a cost in setting these parameters. Of course, if we know the form of the desired solutions, they are relatively easy to estimate – but this is cheating. The performance is sensitive to both parameters, though we have found in tuning experiments that a relatively wide band for each will give reasonable performance. This, of course, is a limitation of our current incremental process,

We hypothesize that the primary issue with the more difficult problem, is that *DEVTAG* has no way to directly represent the required synchrony between the $(1/n)$ and $\sin(n*X)$ terms in the formula. (This might be viewed as analogous to the problem in biological development, of roughly synchronizing development rates for such organs as the lens and ball of the eye, so that the eye is able to focus).

The work reported here is primarily a pilot study for a larger-scale approach with a more sophisticated developmental process. The *TAG* representation is crucial, because it removes any difficulty in ensuring that intermediate developmental stages can be evaluated. We are replacing *DEVTAG*'s trivial developmental process with a more sophisticated approach based on a *TAG* analogue to L-systems (currently in debugging).

We hope that this system may be able to solve the first difficulty (extra parameters) through self-adaptation. While we will provide a fixed evaluation schedule to the L-system, the system itself will be free to adapt the amount of change from evaluation point to evaluation point (unlike the current approach). We think it may not be necessary to pre-set this.

The more flexible representation should also be able to represent more complex building blocks, and in particular, to represent the synchronization of the $1/n$ and $\sin nX$ terms directly. Of course, being able to represent a building block doesn't necessarily imply being able to learn it – but it is a necessary precondition. Optimistically, the system might be able to learn a generalized "betaN" building block, and then specialize it as needed to beta1, beta2,..

We note that the requirement to supply a *family* of problems of increasing difficulty to our system may limit its application. The *DEVTAG* method creates an incremental problem solver; it must solve a problem in stages, rather than

solving a whole problem at once. We would argue that this is a necessary property of a powerful problem solver, but accept that it limits the range of applicable problems. However we note that such families of problems arise in many important problems. Most notably, the generalization hierarchy often provides such families in practical real-world problems. It is often desirable to find a simple, general model that fits a large dataset reasonably well, and specialize it to more specific and complex models that fit specific regions of the dataset more accurately. We aim to identify a range of such problems, both toy and real-world, and apply the DEVTAG approach to them.

Finally, our primary aim with this approach is to develop systems which are able to emergently develop modular structure, as in biological evolution. This requires developing metrics to measure modularity. Hornby [10] has developed such metrics, but unfortunately they rely on specific representation of modularity, hence are not suitable for measuring emergent modularity. An alternative is to use compression-based metrics. A modular structure has repeated sub-structure; hence it should be more compressible. Thus it should be possible to use compression-based metrics to indirectly measure modularity. Software to do this is currently under development. However the problem is complex. Ineffective code (bloat) typically incorporates a large amount of repeated code. Thus the modularity metrics should measure the compressibility of the effective code only (otherwise, it may just measure the amount of bloat – modularity in ineffective code is not very interesting). Hence measuring modularity also requires us to develop reliable methods of removing ineffective code. While this is, in general, a Turing-incomplete problem, we hope to report soon on some methods which, for symbolic regression problems, do seem to eliminate the vast bulk of the ineffective code.

REFERENCES

- [1] Nguyen, Xuan Hoai, McKay, R. I. and Abbass, H. A.: Tree Adjoining Grammars, Language Bias, and Genetic Programming. In Ryan, C., Soule, T., Keijzer, M., Tsang, E. P. K., Poli, R. and Costa, E. (eds): Proceedings of EuroGP2003, Lecture Notes in Computer Science, Vol. 2610, Springer-Verlag (2003), pp. 335-344.
- [2] Koza John R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA, 1992.
- [3] Angeline, P.J. "Evolutionary Algorithms and Emergent Intelligence", PhD thesis, Computer Science Department, Ohio State University, 1994.
- [4] Koza. John R. Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, Cambridge Massachusetts, May 1994.
- [5] Rosca, Justinian P. and Ballard, Dana H.: Hierarchical Self-Organization in Genetic Programming. In Rouveirol, C. and Sebag, M. (eds): Proceedings of the Eleventh International Conference on Machine Learning, Morgan Kaufmann, 1994.
- [6] Haddow, P.C., Tufté G., and van Remortel P.: Shrinking the genotype: L-systems for Evolvable Hardware. In Liu, Y., Tanaka, K., Iwata, M., Higuchi, T. and Yasunaga, M. (eds): Evolvable Systems: From Biology to Hardware, 4th International Conference, ICES 2001, Lecture Notes in Computer Science, Vol. 2210, Springer-Verlag, Berlin – Heidelberg – New York (2001) 128 –139.
- [7] Miller J.F and Thomson, P.: A Incremental Method for Growing Graphs and Circuits. In Tyrrell, A.M., Haddow, P.C. and Torresen, J. (eds): 4th International Conference on Evolvable Systems: From Biology to Hardware, LNCS 2210, Springer-Verlag (2003), 93 – 104.
- [8] Joshi, A.K., Levy, L. S., and Takahashi, M.: Tree adjunct grammars, *Journal of Computer and System Sciences*, 21(2) (1975), 136 – 163.
- [9] Nguyen Xuan Hoai, McKay, R.I., Essam, D.L. and Chau, R.: Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming: The Comparative Results.. In Yao, X. (ed): Congress on Evolutionary Computation (CEC2002), IEEE Press, 2002, vol. 2, 1326-1331.
- [10] Hornby, Gregory S.: Measuring, Enabling and Comparing Modularity, Regularity and Hierarchy in Evolutionary Design. In Beyer, H.-G., O'Reilly, M., Arnold, D. V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E., Lipson, H., Llorca, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P. and Zitzler, E.: Proceedings of the 2005 Genetic and Evolutionary Computation Conference (GECCO'05) ACM Press (2005) Vol.2, 1729-1736
- [11] McKay, R.I.(Bob) Hoang, Tuan Hao, Essam Daryl, Nguyen Xuan Hoai: Incremental Evaluation Genetic Programming: The preliminary results EuroGP2006, Lecture Notes in Computer Science (LNCS), vol. 3905, 280-289, Springer-Verlag, 2006.
- [12] Nguyen Xuan Hoai, McKay, R I, Essam, D L and Hoang Tuan Hao, 2005, Genetic transposition in tree-adjointing grammar guided genetic programming: the duplication operator. In *Proceedings of the 8th European Conference on Genetic Programming (EuroGP2005)*, Springer-Verlag.