Developmental Evaluation in Genetic Programming: A TAG-Based Framework

Tuan-Hao Hoang[1], Daryl Essam[1], McKay RI (Bob)[2], Nguyen Xuan Hoai[3]

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

[2] School of Computer Science & Engineering, College of Engineering, Seoul National University, Korea
Email: rim@cse.snu.ac.kr

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

Abstract. We build on our previous feasibility studies [16, 17], which demonstrated the impact of evaluation during development in the DEVTAG system, and here present a full-fledged developmental system DTAG3P, with developmental evaluation, based on Tree-Adjoining Grammars (TAG). While DEVTAG used only a trivial developmental process, DTAG3P uses L-systems to encode TAG derivation trees, the L-systems permitting a full developmental process. DEVTAG was previously shown to dramatically out-perform standard Genetic Programming (GP) on some structured families of problems; here, we examine DTAG3P's performance on one of these families, and find a further major increment in performance over DEVTAG. DTAG3P achieves this despite dispensing with two extra control parameters which it was necessary to introduce into DEVTAG.

1 Introduction

Genetic Programming (GP) was developed by Koza [2] in 1992. Based on observations of biological systems, it uses an abstraction of Darwin's natural selection mechanisms to evolve populations of solutions to problems. However unlike biological systems, it is not very good at finding structured solutions, rarely finding any hierarchical or modular structure, and exhibiting relatively poor re-use. By contrast, hierarchical and repeated structures are widespread in biological systems, with the homeobox gene complex being perhaps the best-known example outside the biological community [19]. A general method to generate hierarchical, modular structures would potentially improve both the scalability and the adaptability of GP solutions, with consequent widening of the application of GP techniques.

There has been a wide range of approaches to solving this problem. For example, Angeline [10] developed a technique called Module Acquisition, 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) [3], which is probably the most popular modularization method used in GP. Rosca investigated an Adaptive Representation [4], which is based on the discovery of useful building blocks of code. This approach greatly improved search efficiency on the problems considered. However, all these techniques are imposed on the system by programmer intervention, rather than arising as a natural consequence of the evolutionary behaviour of the system; moreover, none has so far demonstrated the scale of modularization and hierarchical organization apparent in biological systems.

A number of authors have built evolutionary developmental systems using Lindenmayer (L) Systems. Jacob [14] investigated Genetic L-System Programming using context-free L-systems (OL) with stacking capability and an evolutionary algorithm to learn L-systems for the creation and development of artificial flowers. Haddow et al [15] used L systems for digital circuit design, while Hornby et al [11-13] used them to evolve generative design specification that could create more complex modules from simpler ones.

Nevertheless, modular structure has not been clearly demonstrated in existing developmental GP systems. We have argued in [16-17] that this is a consequence of the single-evaluation used in these systems; that modularity provides no advantage for an individual which is only evaluated once, hence evolution finds it difficult to select for. However if an individual is evaluated at different stages of development, then modularity can provide an advantage to the individual, not just the population, and hence can be readily selected for. We argued that this is why modularity is so ubiquitous in natural systems in which the individuals are evaluated multiple times on problems of increasing difficulty throughout the developmental process, as occurs in higher animals (more speculatively, it might also explain why modularity is much less marked in the genotypes of lower organisms, and especially of prokaryotes).

As a pilot, in order to test the above hypothesis, in [16-17], we implemented incremental evaluation in a new grammar-guided genetic programming system called DEVTAG. The preliminary results on our chosen symbolic regression problems were promising. However DEVTAG undergoes a trivial developmental process, analogous to the development of undifferentiated colony species such as sponges (more precisely, as implemented, there is no development at all, though the system is logically equivalent to one with a simple developmental process). Its 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. Moreover, 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 pre-fixed parameters, even if we know the form of the desired solutions, although reasonable values are relatively easy to estimate. This paper represents the second phase of our project: having confirmed through our pilot study the benefits of developmental evaluation, we now introduce a new sophisticated developmental process using a new representation, Developmental Tree Adjoining Grammar Guided GP (DTAG3P), which uses L-systems to encode tree adjoining grammar guided (TAG) derivation trees. We applied our new Grammar Guided GP (GGGP) system on the previously-studied symbolic regression family of target functions and compared the results with those in [16-17] of DEVTAG, TAG3P, and GP.

The paper is therefore organised as follows. The next section briefly describes TAGs and TAG based Genetic Programming (TAG3P). Section 3 briefly reviews our previous incremental evaluation based on TAG3P, and the resultant DEVTAG system. Section 4 introduces L-systems, and our developmental approach to TAG based L-system and Developmental Tree Adjoining Grammar Guided Genetic Programming (DTAG3P). Experimental setups are described in section 5. Section 6 provides the results and discussions. Conclusions and future work are laid out in the last section.

2 Tree Adjoining Grammar, TAG Based Genetic Programming

The following section gives a brief, somewhat intuitive introduction to TAG; a fuller description of TAG may be found in [1].

2.1 Tree Adjoining Grammars (TAGs)

TAGs are tree-generating and analysis systems, first proposed by Joshi [5-6] for Natural Language Processing (NLP) purposes.

TAG aims to more directly represent the structure of natural languages than is possible in Chomsky languages, especially the process by which natural language sentences can be built up from a relatively small set of basic linguistic units using insertable sub-structures. Thus 'The cat sat on the mat' becomes 'The black cat sat lazily on the mat' by the subsequent insertion of the elements 'black', and 'lazily'. For more detail see [18]. A tree-adjoining grammar comprises of terminal symbols, non-terminal symbols, a start symbol, initial trees called α trees and auxiliary trees indicated β trees. Figure 4 show the trees used in this paper. All individuals are composed of instances of these trees – a diverse group of operators has been provided so that a variety of solutions might be explored.

The key operation used with TAG is adjunction. Adjunction builds a new tree γ from an auxiliary tree β and a tree α by inserting β into α at a specified place. Adjunction is illustrated in Figure 1. This process may be formalized as described in [18].

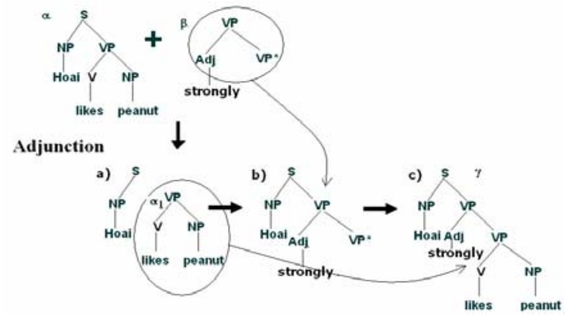

Fig. 1. *An example of the Adjunction operator*

2.2 TAG Based Genetic Programming

Tree Adjoining Grammar Guided Genetic Programming (TAG3P) [1] is a GGGP system, a typical only in the use of TAG derivation trees, rather than CFG derivation trees, as the evolutionary representation. However this small difference leads to a crucial new feasibility property: any rooted sub-tree of a TAG derivation tree is also a valid TAG tree. As a result, one can stop at any time in growing a derivation tree, and still have a valid tree. For example, if a derivation tree consisted of β_1 adjoined to α (both from figure 4), we could either stop at α before considering β_1, generating the derived tree X, or consider the entire tree and generate X+X.

3 Incremental Evaluation Based On TAG3P

Our developmental system (DEVTAG) in [16-17] uses incremental evaluation based on TAG3P to evaluate fitness. The problem chosen for investigating our system is the symbolic regression problem [7] with its increasing difficulty of polynomial degree as target functions. We expect to be able to exploit this increasing difficulty using our new representation and GP system. Our description below is adapted from [16].

To fulfill the requirement of tackling increasingly difficult problems throughout development, the individual is separated into layers corresponding to the stages of the developmental process of L-systems. 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. For the particular problem family considered in [16-17], the program tree might be divided as follows:

Stage 1 for function $F_1 = X$
Stage 2 for function $F_2 = X^2+X$
…
Stage 9 for function $F_9 = X^9+X^8+X^7+X^6+X^5+X^4+X^3+X^2+X$.

We note that $F_{i+1}=F_i*X+X$ (with i=1,2,..8); that is, the family of problems *can* be solved incrementally. We used tournament selection, which only requires a fitness ordering of individuals. For DEVTAG, we use a special multi-stage comparison to generate this ordering. Corresponding to the 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 $|F(I_1, i) - F(I_2, i)| < \varepsilon$
        i := i + 1;
IF (F($I_1$, i) < F($I_2$, i)) THEN $I_1$ wins ELSE $I_2$ 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… , 20.35), and I_2 (10.06, 14.66, … , 10.35). In this case, I_2 would be chosen for evolution.

The individual representation and operators in DEVTAG are the same as in TAG3P. Schematically, we could summarise the relationship as:

DEVTAG = TAG3P representation/operators + Incremental Evaluation.

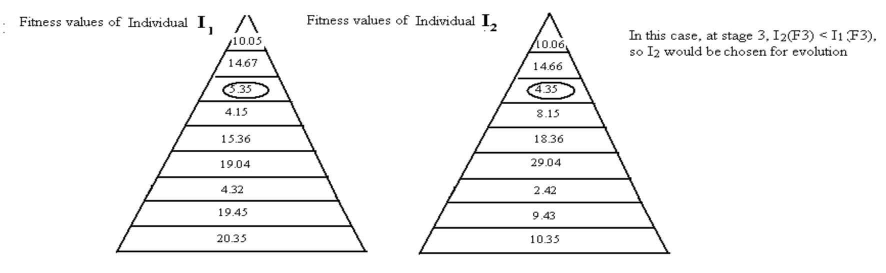
Fitness values of Individual I1
10.05
14.67
5.35
4.15
15.36
19.04
4.32
19.45
20.35
Fitness values of Individual I2
10.06
14.66
4.35
8.15
18.36
29.04
2.42
9.43
10.35
In this case, at stage 3, I2(F3) < I1(F3),
so I2 would be chosen for evolution

Fig. 2. *An example of comparing two individual in DEVTAG*

4 A New Approach To TAG-Based Developmental GP

In this section, we first briefly describe L-systems, then detail our new approach (DTAG3P) on developmental genetic programming using a new representation called TAG based L-systems (L-TAG).

4.1 L-Systems

L-Systems were introduced by Lindenmayer in 1968 [8], using the central concept of a rewriting mechanism. The essential difference between Chomsky grammars and L-systems lies in the method of applying productions. In Chomsky grammars, productions are applied non-deterministically, whereas in simple L-systems they are applied in parallel, and simultaneously replace all letters in a given word. This difference reflects the biological motivation of L-systems, providing a commonly used formalism to describe developmental processes of natural organisms. The detailed definition of L-systems and their operation are given in [9].

4.2 TAG Based L-systems

An L-systems in our representation comprises of a triple G (V, ω, P), where V is the set of alphabet, or set of predecessors $\{L_1,L_2, L_3,\ldots\}$ (note that predecessors are actually auxiliary trees (see β_{13}-β_{16} trees from Fig. 4), this notation is merely indicative of a pre-order traversal of those trees). The initial axiom ω is an initial tree adjoined to a letter from the alphabet (a predecessor). The set of rewriting rules $P = \{P_i: i = 1..m\}$ have the form P_i: $L_i \rightarrow S_1S_2\ldots Sn$, where S_i is either a β-tree or a predecessor. For example, G'(V', ω', P') denotes an L-system with $V'=\{L_1,L_2,L_3,L_4\}$, $\omega'=(\alpha_1,L_1)$, and P' is as below:

P'$_1$: $L_1 \rightarrow \alpha_1\ \beta_1\ \beta_2\ L_2\ \beta_3 L_4$
P'$_2$: $L_2 \rightarrow \alpha_2\ \beta_2\ \beta_1\beta_4\ \beta_3 L_2 L_3$
P'$_3$: $L_3 \rightarrow \alpha_1\ \beta_5\ \beta_6\ L_4\ \beta_7\ \beta_8 L_1$
P'$_4$: $L_4 \rightarrow \alpha_2\ \beta_1\ L_2\ \beta_4\ \beta_7 L_3$

Figure 3 depicts a TAG derivation tree which may be generated from the L-system above. It starts with a random initial tree, α_1 tree, and letter L_1. L_1 is then replaced by its successor. This successor has two letters L_2, L_4, which are then expanded by their successors. This process is repeated until the number of developmental phases, which was specified for the particular problem, is reached. In this paper, for the family of the polynomial symbolic regressions (F_1, F_2, ..,F_9), the developmental phases stop after DEPTH=9 times repetitions.

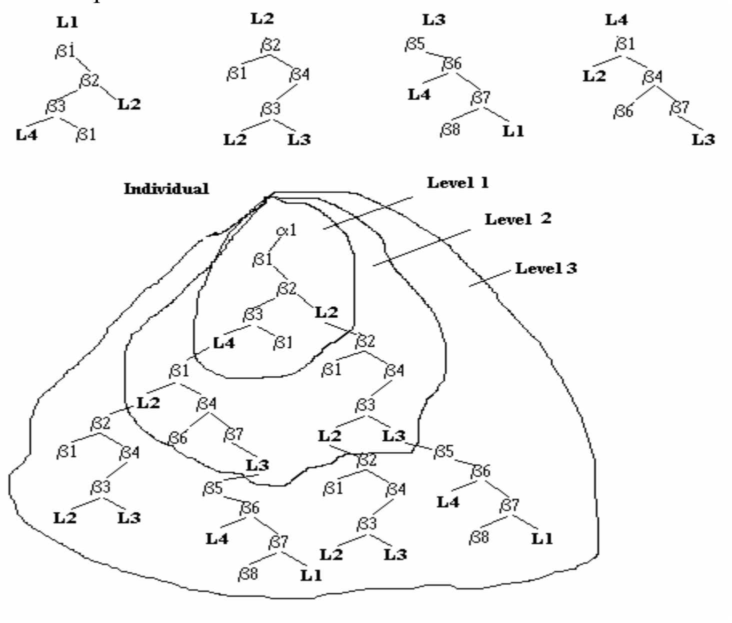

Fig. 3. *An example of TAG based L-systems*

4.3 Developmental Tree Adjoining Grammar Guided Genetic Programming

DTAG3P uses L-TAG systems to encode Tree Adjoining Grammars and thus set the language bias for a genetic programming system. As in Koza-style GP, DTAG3P consists of five main components:

Initialization procedure: The process starts by randomly generating L-systems, using a fixed parameter NUMRULES, the number of L-system rules. V is a set of letters $\{L_1, L_2, \ldots, L_{NUMRULES}\}$ (considered as β-trees from the auxiliary tree set in G_{lex}, see Fig.4), then axiom ω is generated by selecting a random α tree from the initial tree set in G_{lex}, adjoined to randomly chosen one letter from the alphabet. The successor of each rule is subsequently augmented by β-trees randomly drawn from the auxiliary tree set in G_{lex} by using adjunction at random places until the chosen number of β-trees (randomly chosen from MINBETAS to MAXBETAS) is reached.

After that, TAG derivation trees are produced by decoding the L-systems generated. A parameter DEPTH is used to specify the number of cycles of replacement of a letter by its successor.

Genetic operators:

- *Crossover*: Crossover is a two step process. First, rules are altered, with a 50% chance of swapping successors of the same randomly chosen predecessor of two parents, and a 50% chance of copying a random rule from a parent to a child. After that, a normal sub-tree crossover operator occurs in a random successor of a parent. If no crossover occurs, copy the parents to the new children.
- *Mutation*: Mutation operator occurs in a random successor as a sub-tree mutation operator in TAG3P.

Parameters: The minimum and maximum size of genomes (MIN_SIZE, MAX_SIZE), size of population (POP_SIZE), maximum number of generations (MAX_GEN), number of rules (NUMRULES), the minimum and maximum number of β-trees (MINBETAS, MAXBETAS) in a successor and probabilities for genetic operators.

Selection methodology and reproduction: These are as in typical evolutionary algorithms. In this paper, tournament selection is used.

Fitness evaluation: DTAG3P uses the incremental evaluation described in the previous section to evaluate fitness.

5 Experimental Setup

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 set is X.

Formally:

G = (N,T,P,S}

S = EXP – the start symbol

N = {EXP, PRE, OP, VAR, CONST}

T = {x, sin, cos, lg, ep, +, -, *, /},

(ep is exponential, lg is log function).

P consists of

EXP $\rightarrow$ EXP OP EXP | PRE EXP | VAR | CONST

OP $\rightarrow$ + | - | * | /

PRE $\rightarrow$ sin | cos | lg | ep

VAR $\rightarrow$ x

As from [18], this generates a corresponding TAG G_{lex}= {N={EXP, PRE, OP,VAR},T={X, sin, cos, log, ep,+, -, *, /, (,)}, I, A) where I$\cup$ A is as in Figure 4.

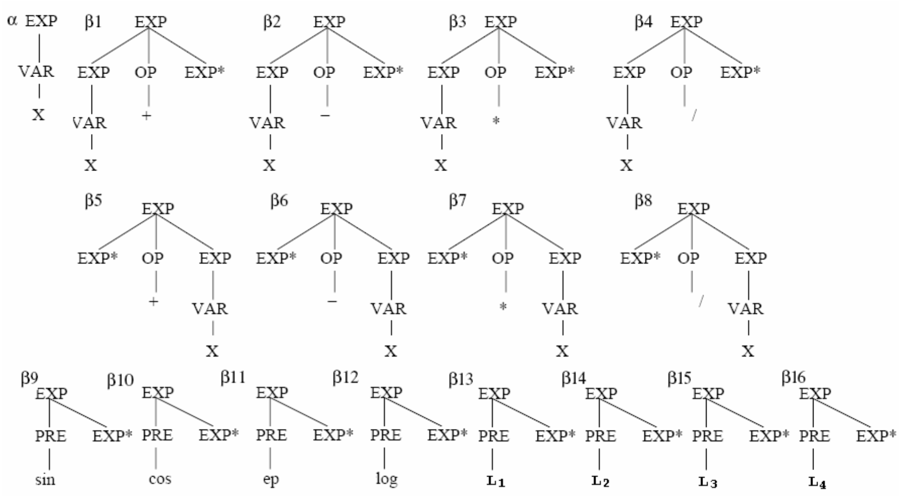

Fig. 4. *Elementary trees for G_{lex}*

Table 1. Parameter settings for the symbolic regression problem

Objective	Find a symbolic regression function F_9(GP, TAG) or F_1, F_2, F_3, .. ,F_9(DEVTAG, DTAG3P) that exactly fits a given sample of 20 (xi, yi) data points.
Success Predicate	Sum of errors over 20 points < $\varepsilon = 0.01$
Terminal sets	X - the independent variable
Operators(Function set)	+,-,*,/, sin, cos, exp, log
Fitness Cases	The sample of 20 points in the interval [-1..+1].
Fitness	Sum of the errors over 20 fitness cases.
Genetic Operators	Tournament selection(3), crossover between rules, sub-tree crossovers and sub-tree mutation on sucessors on DTAG3P, sub-tree crossovers and sub-tree mutations using on TAG3P, normal standard crossovers and muations using on GP
Parameters	The crossover probability is 0.9. The mutation probability is 0.1.
Min/Max initial zise on DTAG3P, TAG3P	2 to 1000
Max depth using for GP	20
Number of rules	4
Min/Max number of β-tree in each successor	1 to 5
Population size	100,250,500,1000

Some preliminary results using our incremental evaluation method on this incremental problem family were presented in [16]. There, we reported that, with the same number of function evaluations, DEVTAG substantially outperformed both Koza-style 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. To investigate the effect of the full developmental approach of DTAG3P, four experimental settings have been used by changing different population sizes (POPSIZE = 100, 250, 500 and 1000), with the maximum generation size (MAXGEN) changing correspondingly to keep a constant budget of 229,500 (9x51x500) function evaluations; these are to the same experimental settings as in [16]:

6 Results and Discussion

Table 2. Successful runs (from 100 runs)

POPSIZE	**100**	**250**	**500**	**1000**	**Statistical Significance**
DTAG3P	41	72	73	78	α=0.01
DEVTAG	13	33	27	3	α=0.01
TAG	3	8	9	4	α=0.01
GP	0	0	0	0	α=0.01

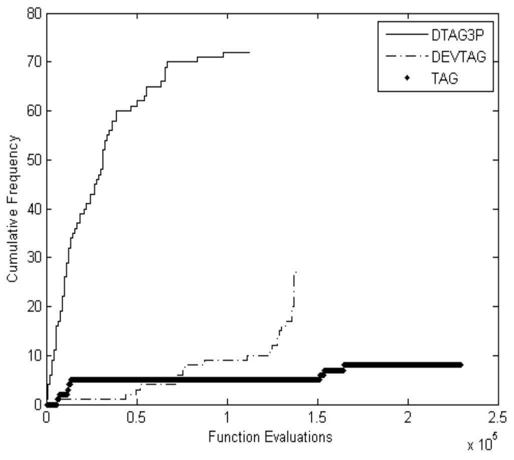

Fig. 5. *Cumulative success frequency of DTAG3P, DEVTAG and TAG vs. number of function evaluations*

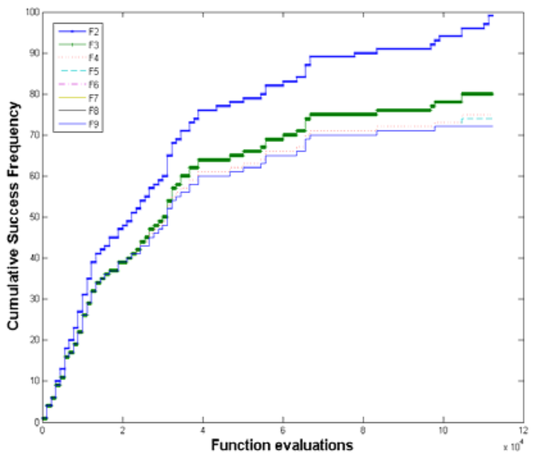

Fig. 6. *Cumulative success frequency of DTAG3P on each of the 9 problems*

Table 2 shows the number of successful runs, which found the exact solutions, out of 100 for each of the four treatments and four different population sizes. It is clear that DTAG3P very substantially outperforms the other representations at all population size settings. For example, for the POPSIZE=250 setting, DETAG3P's success was 72% which is well above that achieved from other treatments; DEVTAG's success: 33%, TAG: 8%.GP: 0 – it means that the GP runs were never successful.

Figure 5 shows the cumulative probability of success of the three successful treatments (for population size 250), plotted against the number of function evaluations used in the evolution. The steepest and shortest curve has the lowest computational costs in terms of number of function evaluation. To help in understanding how DTAG3P incrementally solves the problems, figure 6 shows the cumulative probability of success of DTAG3P, for all 9 symbolic regression problems, for the particular case of population size 250. It is worth nothing that DTAG3P gives us solutions to all the other eight functions, at no extra computation cost.

From table 2, as with DEVTAG, DTAG3P, which uses the same incremental evaluation method as DEVTAG, is very effective at finding extract or near-exact solutions to the problem at all population size settings. However, DTAG3P's performance is much better than DEVTAG. We interpret this as resulting from DTAG3P's more flexible representation. Because of L-system self-adaptation, even though we provide a fixed evaluation schedule, the system is free to adapt the amount of change from evaluation point to evaluation point (unlike DEVTAG, with its pre-determined initial depth and increment), allowing DTAG3P to find solutions more easily.

From figure 5, we see that it takes DEVTAG some time to find solutions at all, but once it does so, it rapidly finds more. We interpret this as DEVTAG needing a number of evaluations to get evolution running well at the lower levels, but once it does, solutions to F9 follow rapidly. DTAG3P find solutions even faster and better as a result of its re-rewriting mechanism. The patterns found in previous levels are copied to the next levels, saving time by solving a complex problem built on simpler ones. We note in passing that counting function evaluations under-estimates the computational advantages of DEVTAG and DTAG3P. Because of the stepped evaluation method, many of the evaluations of higher order functions are actually never used in selection; in a system focused on performance rather than research, they would not be evaluated at all (this point was studied in greater detail in [17]).

Figure 6 appears to confirm this interpretation, of gradually finding lower-level solutions, with the solutions of higher complexity following fairly rapidly. There is a strong suggestion from the very closeness of the curves, that once DTAG3P has found building blocks for lower-level solutions, they are quickly assembled in forming the higher-level solutions. By using the rewriting grammar mechanism, we believe DTAG3P is achieving this by replicating building blocks and creating modularity. We are currently in the process of implementing a method, based on tree-compression methods, for measuring this directly. At the very least, the results strongly support the view that incremental learning of a family of increasingly difficult functions has been demonstrated.

To investigate whether DTAG3P's performance subjective to the two new parameters: number of rules (NUMRULES), and the minimum/maximum number of β-trees (MINBETAS, MAXBETAS) in a successor, we did another experiment with a population of 250 and 100 generations. The number of runs was 30. The similar (not statistically significant difference) performance of DTAG3P with different settings for these parameters indicates that, on the problems tried, these parameters do not significantly affect DTAG3P's ability to find exact solutions.

Table 3. Successful runs (from 30 runs, change NUMRULES, keep other parameters the same)

NUMRULES	**2**	**4**	**6**
DTAG3P	76.6	70	83.3

Table 4. Successful runs (from 30 runs, change MAXBETAS, keep other parameters the same)

MAXBETAS	**3**	**5**	**7**
DTAG3P	60	70	66.6

7 Conclusions and Future Works

The results strongly suggest that the DTAG3P approach, using a TAG-based analogue of L-systems rewriting rules, support the hypothesis that evaluation during development, on a family of problems of increasing difficult, can lead to incremental learning (and also, to modular solutions – this is certainly our impression on viewing the evolved genotypes).

The computational cost of the approach is also worth noting (though it is not the primary focus of this work), DTAG3P being much less expensive than the other approaches in computational cost, as well as yielding much more (a family of functions rather than just one) in return for that computational investment.

In the near future, we aim to apply this system to a range of problem families, and to analyse its behaviour, particularly in terms of the modularity and complexity of evolved solutions.

Acknowledgement

The fouth author was partly funded by a national research grant on fundamental sciences, grant number - #203106, for doing this work.

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. (editors): Proceedings of EuroGP2003, LNCS, Vol. 2610, pp. 335-344, Essex, Springer-Verlag, 2003.
2. Koza John R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA, USA,1992.Angeline, P.J "Evolutionary Algorithms and Emergent Intelligence", PhD thesis, Computer Science Department, Ohio State University, 1994.
3. Koza. John R. Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press, Cambridge Massachusetts, May 1994.
4. 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.

5. Joshi, A.K., Levy, L. S., and Takahashi, M.: Tree adjunct grammars, *Journal of Computer and System Sciences*, 21(2), Pages 136 – 163, 1975.
6. Joshi A.K and Y.Schabes, Tree Adjoining Grammars, in G. Rozenberg and A. Saloma, editors, Handbook of Formal Languages, Springer-Verlag, 69-123,1997.
7. 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, vol. 2, 1326-1331, 2002.
8. A Lindenmayer. Mathematical models for cellular interaction in development parts I and II. *Journal of Theoretical Biology*, 18:280-299 and 300-315, 1968.
9. Prusinkiewicz, P., Lindenmayer, A.: The Algorithmic Beauty of Plants. Springer-Verlag (1990).
10. Angeline, P. J. Pollack, J. Evolutionary Module Acquisition, Proceedings of the 2nd Annual Conference on Evolutionary Programming, pp. 154-163, MIT Press, 1993.
11. Hornby, Gregory S.: Measuring, Enabling and Comparing Modularity, Regularity and Hierarchy in Evolutionary Design. In Beyer, H.-G. et al.: Proceedings of the 2005 Genetic and Evolutionary Computation Conference, ACM Press (2005) Vol.2, 1729-1736.
12. Hornby, G. S. and Pollack, Jordan. B. Evolving L-Systems To Generate Virtual Creatures Computers and Graphics, 25:6, pp 1041-1048. 2001.
13. Hornby, G. S. Generative Representations for Evolving Families of Designs. Genetic and Evolutionary Computation Conference, pp 1678-1689, Springer-Verlag, 2003
14. Jacob, C.: Genetic L-system Programming. In Davidor, Y., Schwefel, P., eds.: Parallel Problem Solving from Nature III, Lecture Notes in Computer Science. Vol. 866. (1994) 334-343.
15. Haddow, P.C., Tufte 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.
16. R.I. McKay, Tuan-Hao Hoang, D. Essam, and Nguyen Xuan Hoai, Developmental Evaluation in Genetic Programming: The Preliminary Results, EuroGP 2006, Lecture Notes in Computer Science (LNCS), vol. 3905, 280-289, Springer-Verlag, 2006.
17. Tuan-Hao Hoang, Daryl Essam, R.I(Bob) McKay and Xuan Hoai Nguyen, Solving Symbolic Regression Problems using Incremental Evaluation in Genetic Programming, to appear in Proceedings of IEEE Congress on Evolutionary Computation, IEEE Press, 2006.
18. Nguyen Xuan Hoai, R.I McKay and D. Essam. Representation and Structure Difficulty in Genetic Programming, IEEE Trans on Evolutionary Computation, 10(2), 2006, 157-166.
19. Gerhard Schlosser and Gunter P. Wagner (eds.) Modularity in Development and Evolution, The University of Chicago Press, 2004.