Developmental Evaluation in Genetic Programming: A Position Paper

Tuan Hao Hoang, *Member, IEEE,* R I (Bob) McKay, *Senior Member, IEEE,* Daryl Essam, *Member, IEEE,* and Xuan Hoai Nguyen, *Member, IEEE*

Abstract—Standard genetic programming genotypes are normally highly disorganized, poorly structured and not normally re-used (i.e, bloat etc.). This is also true of previous developmental genetic programming systems [1], [2], which exploit regularity by using procedures, functional modules, or macros and parameters passing. Instead, in biological developmental evolution, nature works through code duplication to generate modularity, regularity and hierarchy. Moreover, previous developmental approaches have only one level of evaluation for each individual - an approach that only has adaptive advantages to entire species, rather than particular individuals. We argued in [3], [4] that evaluation during development is necessary for structural regularity to emerge. To confirm the benefits of developmental evaluation and the contribution of code duplication to nature, our new sophisticated developmental process uses a new representation. That system, Developmental Tree Adjoining Grammar Guided GP (DTAG3P)- used L-systems to encode tree adjoining grammar guided (TAG) derivation trees, and has been investigated in [4]. On the family of problems tried, we have found further major improvement in performance over original approaches TAG3P and GP, in ways which suggest structural regularity solutions have been exploited.

Index Terms—Regularity, developmental evaluation, developmental genetic programming.

1 Introduction

Genetic programming (GP) was developed by Koza [5] in 1992 with the idea of getting computers to automatically solve problems without having to tell them how to do it. Based on observations of biological systems, GP uses an abstraction of Darwin's natural selection mechanisms to evolve populations of solutions to problems. One of the biggest problems with GP is called coat bloat. Genetic programming produces solutions with large amounts of unnecessary code that exponentially increases over time and is not proportional to increase the quality of the solutions produced. An analogy to GP, in nature, the human genome has less than 5% of its 3.2 billion As, Cs, Gs and Ts that construct the human genome [6]. The remainder is just rubbish, plain and simple. However, unlike GP, the bloat in Hox gene seems steady and not exponential.

Moreover, unlike biological systems, genetic programming is not very good at finding scalable structured solutions, rarely finding hierarchical or structural regularity, and exhibiting no hierarchical or structural regularity, relatively poor repetition of units, self-similarity, or re-use. By contrast, the natural evolutionary systems on which it is based are able to evolve structural regularity (e.g., the Hox gene complex is a subset of the Homeobox genes which are a larger family of transcription factor proteins and share a similar 60 amino acid DNA binding homeodomain [7]).

There has been wide range of approaches to solving this problem in GP. Koza introduced Automatically Define Functions (ADF) [1], where parts of the program are modularized and encapsulated, this approach also reduced coat bloat. Koza showed [1] that ADFs

• *H. Hoang and D. Essam are with the School of IT and EE, University of New South Wales@ADFA, Canberra, Australia. B. McKay and X. Nguyen are with the School of CS and E, Seoul National University, Korea.*

allows genetic programming systems to better exploit regularities of problem domains, improving system performance. Spector [2] investigated Automatic Defined Macros (ADMs) these were based from ADFs, with also the idea of using the term ”macro” as in many programming languages. In [2], he observed that ADMs, like subroutines, can assist in the modularization of complex programs and in the exploitation of domain regularities. It was suggested that ADMs are likely to be useful for evolving intelligent action systems for complex environments.

However, these approaches are apparently not sufficient. Natural evolution exploits regularities and modules by copying or duplicating. For examples, Moghadam HK et. al [8] showed the evidence of Hox gene duplication in rainbow trout where many duplicated genes appear in the genome and share a high percentage of amino acid similarity with one another. Ohno’s book in 1970, *E*volution by Gene Duplication[9] and its update by Zhang in 2003 [10] clearly demonstrated that the creation of new structures and behaviors of a biological organism begins with a gene duplication. Without gene duplication, the flexibility of genomes or species in adapting to changing environments would be severely limited. In other words, nature does not use procedures or macros and parameters passing, but instead works through code duplication to generate modularity, regularity and hierarchy. We don’t always need to do the same as nature, but we do need to understand why nature can work in a particular way and we can’t (i.e., why nature can generate structural regularity that GP can’t).

Recently, a number of authors have built evolutionary developmental systems in genetic programming. Rosca investigated an Adaptive Representation [11], which is based on the discovery of useful building blocks of code. This approach greatly improved search efficiency on the problems considered. Jacob [12] 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 el al. used Lindenmeyer systems, the mathematical formalism used for biological development study, for digital circuit design [13]. Later, Developmental Cartesian Genetic Programming was investigated by Miller et. al., with attempting to evolve a cell that can construct a larger program by iteration of the cell’s program in its environment [14].

Nevertheless, structural regularity, which is the correlation of patterns within an individual (e.g., the repetition of units, symmetries, self-similarities, smoothness), has not been clearly demonstrated in existing developmental GP systems. The development in GP allows regularity but it does not guarantee it: it has to have an evolutionary benefit if it’s to emerge. We argue that this is because an individual in a GP system is generally expected to solve problems immediately, without the benefit of a developmental phase. Although various morphological systems have been used in some previous developmental genetic programming systems to allow program individuals to ”grow” from simple to more complex forms. For examples, Whitley and Gruau in [15] proposed the technique of a cellular developmental process encoded by a grammar tree to generate the families of Boolean neural networks to solve parity problems. Also, Spector and et. al., in [16] implemented Ontogenetic Programming - Ontogenetic HiGP system that includes program self-modification functions in the genetic programming function sets, thereby allowing evolved program individuals to modify themselves during the course of the run. But if these approaches are used with a single evaluation as in most artificial developmental systems, it 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. In other words, if an individual is uncompetitive at any stage of development, it is unlikely to survive to reproduce)(e.g., A Human does not have to be fit just in adulthood. We must also survive in pregnancy, birth and then

childhood). In simpler organisms, the organism faces similar challenges at each stage of development, but in more complex organisms, the challenges increase in complexity as the individual develops. Moreover, once the genes controlling a particular stage of development have succeeded at a particular stage of development, they tend to become fixed, and to change only slowly. We aim to simulate these processes. There are a number of components to this simulation:

1) *Developmental process* governed by 'genes': for this, we use a Deterministic L-system, in which the grammar simulates the genes, and the L-system development process the growth of the organism
2) *Developmental evaluation*: for this, we require a phenotype which we can always guarantee to evaluate, so we use a tree adjoining grammar guided genetic programming (TAG) representation that has ideal 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 subtrees[17].
3) *Evolutionary process* evolving the L-system
4) *Adaptive variation rates*

2 TAG BASED DOL SYSTEM

For our purposes, DOL systems must not only represent development, but must generate at each stage an individual which can be evaluated. We use TAG representation, as introduced in [17]. Briefly, TAG representation consists of an α tree ('The cat sat on the mat') and instructions for adjoining β trees ('black' or 'which it liked') to form a more complex whole ('The black cat sat on the mat which it liked'). In our representation, the DOL triple $G = (V, \omega, P)$ is mapped to TAG representation by defining ω to consist of an α tree together with a predecessor from P, and each letter $\{L_1, L_2, L_3, \ldots...\} \in V$ to be either a predecessor from P or a β tree. Thus the DOL-rewriting of ω corresponds to adjunction of successive β trees into the initial α tree. For example, assume an L-system $G' = (V', \omega', P')$ with $V' = \{L_1, L_2, L_3, L_4\}$, $\omega' = (\alpha_1, L_1)$, and P' being P'_1: $L_1 \rightarrow \beta_1\beta_2\beta_3 L_4\beta_1 L_2$, P'_2: $L_2 \rightarrow \beta_2\beta_1\beta_4\beta_3 L_2 L_3$, P'_3: $L_3 \rightarrow \beta_5\beta_6 L_4\beta_7\beta_8 L_1$, P'_4: $L_4 \rightarrow \beta_1 L_2\beta_4\beta_6\beta_7 L_3$. Figure shows tree representations of these productions, together with three stages of the expansion of this system into a TAG derivation tree.The expansion starts with the TAG initial tree α_1 together with the predecessor L_1. In the stage 1 expansion, L_1 is replaced by its successor in the production rule P'_1. This successor has two predecessors L_2 and L_4. In the stage 2 expansion, these are replaced by their successors using the corresponding production rules P'_2 and P'_4. This leaves us with four available predecessors – two occurrences each of L_2 and L_3. This process continues until predefined limits on the number of stages are reached.

3 A NEW APPROACH TO TAG BASED - DEVELOPMENTAL 3P (DTAG3P)

DTAG3P uses TAG-based DOL-systems to encode Tree Adjoining Grammars, so delimiting the language of the genetic programming system. It is a developmental form of the earlier TAG3P system, and shares many aspects. We describe these briefly, but refer readers to[17] for detail. We assume a lexicalised TAG (LTAG) grammar G_{lex} defining the sets A of α trees, and B of β trees. We evolve DOL rulesets (each ruleset is an evolutionary individual). The ruleset specifies the development of the individual, generating a TAG derivation tree at each stage s of development. This tree is fitness-evaluated against the corresponding problem P_s from our target family of problems. Evaluation uses the standard conversion, first to the corresponding CFG derivation tree, and then to the expression tree [17]. We follow Koza's specification scheme [5], adapting it to incorporate developmental evaluation:

3.1 Initialisation

We randomly generate max_{pop} DOL-systems, each containing n_{rules} rules $R = \{R_1, R_2, \ldots, R_{n_{rules}}\}$. We denote the predecessors of these rules as $\Lambda = \{L_1, L_2, \ldots, L_{n_{rules}}\}$, so that $V = \Lambda \bigcup A \bigcup B$. We randomly select $\omega = (\alpha, L)$: $\alpha \in A$,

$L \in \Lambda$. We construct the successor (RHS) of R_i by first randomly drawing β-trees from B and assigning them to the RHS of R_i, up to a random limit between $min_{betas}, \ldots max_{betas}$, and then randomly drawing num_{letter} predecessors from V and inserting them into the RHS.

3.2 Developmental and Fitness Evaluation

Each individual undergoes a fixed number max_{life} of developmental stages (corresponding to the size of the problem family). Each individual I is expanded through its development stages. At stage s, this generates a TAG derivation tree I_s of G_{lex} and the corresponding CFG derivation tree $CF(I_s)$ of G and expression $exp(I_s)$. We evaluate $exp(I_s)$ against the corresponding problem, P_s, to get a fitness value $fit(I_s)$.

3.3 Selection

DTAG3P uses a developmental form of tournament selection of size $size_{tourn}$. We first compare the individuals on stage 1 fitness $fit(I_1)$. The fittest individuals are carried to stage 2. This is repeated as necessary with $I_2, \ldots, I_{max_{life}}$; if more than one reach max_{life}, we use random choice. Two individuals I, J are considered equal if $|fit(I_s) - fit(J_s) \leq \delta|$. We use an elite of 1.

3.4 Genetic Operators

Individuals in the next generation are produced with probability p_X by *recombination* and $1 - p_X$ by *alteration*.

- *Recombination* takes two individuals $\{P_1, P_2\}$ and creates two offspring $\{C_1, C_2\}$ by a variant of uniform crossover on rules: a rule in $C_1(C_2)$ is with probability p_{copy} copied from the corresponding rule of $P_1(P_2)$, otherwise is randomly selected from the rules of $P_2(P_1)$.
- *Alteration* consists of three sub-operators acting on RHSs of rules:
  - *internal crossover*. We emphasise that this is an exchange of information between components of an individual, not a recombination operator: subtree crossover is performed between rules
  - *subtree mutation*: subtrees are mutated by subtree mutation
  - *lexical mutation*: the symbol in a node is randomly substituted

  The probability of alteration uses an adaptive alteration rate p_{adapt}, initially set to a high value p_{bad}. When a rule is used in a developmental stage which was used to select the parent, it is reset to a lower value p_{good}. Thus the child is more likely to inherit this rule unchanged.

3.5 Parameters

The maximum number of generations max_{gen}, population size max_{pop}, and recombination (p_X, p_{copy}) and alteration (p_{bad}, p_{good}) rates specify the evolutionary system; the number of rules n_{rules} and min_{betas}, max_{betas}, num_{letter} – respectively minimum and maximum number of β trees and of predecessors in a rule RHS – together with the maximum lifetime max_{life} and minimum difference δ, specify the developmental system.

4 Experimental work

4.1 Problem Domain

Two benchmark problems were chosen for our experiments. The first problem was the polynomial symbolic regression problem with its increasing difficulty of polynomial degree, originally due to Koza [1]: $F_n(X) = X + X^2 + X^3 + ... + X^n$, for various values of *n* n=1...9, and the second problem was the family of odd-*k*-parity problems $\{k = 2...10\}$. We expected to be able to exploit the increasing difficulty of the family of these problems using our new representation and GP system.

4.2 Results

4.2.1 The polynomial symbolic regression problem

On the polynomial symbolic regression problem, we observed the performance of various systems: GP, TAG3P, DEVTAG, DTAGF9all, DTAG3P, DTAG3P+; whereas, DTAGF9all is the DTAG3P system with solving the F9 problem at all stages of development, DTAG3P+ is an update version of the DTAG3P by evolving the fix adaptive mutation rate.

TABLE 1
Percentage Success Rate

GP	TAG	DEVTAG	DTAGF9all	DTAG3P	DTAG3P+
0%	8%	33%	0%	73%	100%

TABLE 2
Percentage Success Rate on 8, 10, 12 parity after $1.25 * 10^8$ Node evaluation (30 runs)

	GP	TAG	DTAG3P+
8 Parity	23%	23%	100%
10 Parity	0%	0%	100%
12 Parity	0%	0%	100%

4.2.2 The parity problems

At the second set of experiments we compare the performance of the adaptive variation system DTAG3P+ with the original TAG3P and GP systems on the scaling up parity problems 8, 10, 12.

5 Conclusion

The conclusion goes here.

Acknowledgments

The authors would like to thank...

References

[1] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. Cambridge Massachusetts: MIT Press, may 1994.

[2] L. Spector, "Simultaneous evolution of programs and their control structures," in *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. Cambridge, MA, USA: MIT Press, 1996, ch. 7, pp. 137–154.

[3] R. I. McKay, T. H. Hoang, D. L. Essam, and X. H. Nguyen, "Developmental evaluation in genetic programming: the preliminary results," in *Proceedings of the 9th European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds., vol. 3905. Budapest, Hungary: Springer, 10 - 12 Apr. 2006, pp. 280–289. [Online]. Available: http://link.springer.de/link/service/series/0558/papers/3905/39050280.pdf

[4] H. T. Hao, D. Essam, R. I. McKay, and X. H. Nguyen, "Developmental evaluation in genetic programming: A TAG-based framework," in *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, T. L. Pham, H. K. Le, and X. H. Nguyen, Eds., Military Technical Academy, Hanoi, VietNam, 2006, pp. 86–97.

[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[6] H. Gee, "Unexpected bits and pieces, special report: the ethics of genetics," feb 2001.

[7] G. Schlosser and G. P. Wagner, Eds., *Modularity in Development and Evolution*. The University of Chicago Press, 2004.

[8] H. Moghadam, R. Danzmann, and M. Ferguson, "Organization of hox clusters in rainbow trout (oncorhynchus mykiss): a tetraploid model species," *Journal of Molecular Evolution*, vol. 61, no. 6, pp. 804–818, Dec 2005.

[9] S. Ohno, Ed., *Evolution by Gene Duplication*. Springer-Verlag, 1970.

[10] J. Zhang, "Evolution by gene duplication: an update," *Trends in Ecology and Evolution*, vol. 18, pp. 292–298, Jun 2003.

[11] J. P. Rosca and D. H. Ballard, "Hierarchical self-organization in genetic programming," in *Proceedings of the Eleventh International Conference on Machine Learning*. Morgan Kaufmann, 1994.

[12] C. Jacob, "Genetic L-system programming," in *Parallel Problem Solving from Nature III*, ser. LNCS, Y. Davidor, H.-P. Schwefel, and R. Männer, Eds., vol. 866. Jerusalem: Springer-Verlag, 9-14 Oct. 1994, pp. 334–343. [Online]. Available: http://www2.informatik.uni-erlangen.de/IMMD-II/Persons/jacob/Publications/GeneticLSystemProgramming.ps.gz

[13] P. Haddow, T. G., and van Remortel P., "Shrinking the genotype: L-systems for evolvable hardware," in *Evolvable Systems: From Biology to Hardware, 4th International Conference, ICES 2001*, ser. Lecture Notes in Computer Science, Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, Eds., vol. 2210. Springer-Verlag, 2001, pp. 128–139.

[14] J. F. Miller and P. Thomson, "A developmental method for growing graphs and circuits," in *Evolvable Systems: From Biology to Hardware, Fifth International Conference, ICES 2003*, ser. LNCS, A. M. Tyrrell, P. C. Haddow, and J. Torresen, Eds., vol. 2606. Trondheim, Norway: Springer-Verlag, 17-20 Mar. 2003, pp. 93–104. [Online]. Available: http://www.elec.york.ac.uk/intsys/users/jfm7/ices2003.pdf

[15] F. Gruau and D. Whitley, "Adding learning to the cellular development process: a comparative study," *Evolutionary Computation*, vol. 1, no. 3, pp. 213–233, 1993.

[16] L. Spector and K. Stoffel, "Ontogenetic programming," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Stanford University, CA, USA: MIT Press, 28–31 Jul. 1996, pp. 394–399.

[17] N. X. Hoai, R. I. B. McKay, and D. Essam, "Representation and structural difficulty in genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 2, pp. 157–166, Apr. 2006. [Online]. Available: http://sc.snu.ac.kr/courses/2006/fall/pg/aai/GP/nguyen/Structdiff

PLACE PHOTO HERE

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.