On Synergistic Interactions between Evolution, Development and Layered Learning

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

***Abstract*—We investigate interactions between evolution, development and lifelong layered learning in a combination we call Evolutionary Developmental Evaluation (EDE), using a specific implementation, Developmental Tree-Adjoining Grammar Guided GP (DTAG3P). The approach is consistent with the process of biological evolution and development in higher animals and plants, and is justifiable from the perspective of learning theory. In experiments, the combination is synergistic, out-performing algorithms using only some of these mechanisms. It is able to solve GP problems that lie well beyond the scaling capabilities of standard GP. The solutions it finds are simple, succinct, and highly structured. We conclude the paper with a number of proposals for further extension of EDE systems.**

***Index Terms*—Genetic programming, developmental, evaluation, structural, regularity, modularity, incremental evolution, layered learning**

I. Introduction

EVOLUTIONARY Developmental Systems (EDS) [1] are powerful methods for generating flexible solutions to complex problems. We examine their combination with layered learning strategies, showing that the resulting combination is both biologically plausible and computationally powerful, generating structured and scalable solutions to difficult function-learning problems.

Evolutionary Computation (EC) is based on analogy between computational and biological systems. The analogy is rooted in the insight of Charles Darwin [2] that variation and natural selection can together explain the vast variety of species we see in the natural world.

Any analogy depends on an abstraction, ignoring aspects that are unimportant to the target. The core problem is to determine which aspects are inessential. In the case of EC, it is clear that we have not yet captured important elements of biological evolutionary systems. We cannot evolve an artificial system with the intellectual performance and effectiveness of a human – or even a fly. Some aspects of biological evolution may be important to the performance of any evolutionary system, whether natural or artificial. Others may be specific to the requirements of DNA and protein chemistry, the requirements of survival in a 3-dimensional world, and so forth. Identifying the former, and especially, finding synergies between them, is an important task in progressing EC.

We concentrate on four aspects of evolutionary development in complex organisms: an underlying evolutionary mechanism, a developmental mechanism, multiple evaluations throughout development, and layered learning of increasingly complex problems. Implementing them in a Genetic Programming (GP) system, we investigate their performance, in particular investigating whether they interact synergistically to solve more complex problems than the individual components can handle. The system, DTAG3P, is based on TAG3P, a grammar-guided GP system using Tree Adjoining Grammars (TAG) [3].

In Section II, we discuss the interaction of evolution and development, both in biological systems and in previous work on artificial systems, at the same time introducing the idea of layered learning. Section III introduces the general issue of structure and regularity in both artificial and biological systems, and its relationship to evolution and development, and especially to their interaction. Section IV introduces some necessary technical background in L-systems, TAG grammars and grammar-based GP, together with a brief introduction to compression and its relevance to measuring complexity and regularity. Section V details the DTAG3P system which is the subject of this paper. The experimental approach which we used to validate some of the ideas, and the problem families we used, are outlined in Section VI, with Section VII detailing the relative performance of the different systems in solving these problems. Section VIII looks in more detail at the results, and in particular at the simplicity and regularity of the solutions that are found. We examine the overall capabilities of the DTAG3P system, and the assumptions and limitations of the current stage of this work, in Section IX, leading into a discussion of some of the future research that can follow. We round out the paper in Section X with a summary of the results, and overall conclusions that can be drawn.

II. Evolutionary Developmental Systems

The best-known EC variants [4]–[6] assume an identity or direct matching between phenotype and genotype, corresponding to a posited RNA world, that occured very early in the evolution of life [7]. More recent variants provide for a genotype-phenotype mapping [8]–[10], corresponding to the decoding of DNA to protein in simple unicellular organisms.

While unicellular organisms can optimise themselves for a complex world, they attain only a bounded complexity; the complex individuals of today's biology only began to emerge

Tuan-Hao Hoang is with Le Quy Don Technical University, Hanoi, VietNam
Bob McKay (corresponding author) is with Seoul National University, Korea, email rimsnucse@gmail.com
Daryl Essam is with the University of New South Wales @ ADFA, Australia
Nguyen Xuan Hoai is with Hanoi University, VietNam

once multicellular organisms with developmental mechanisms arose. This insight led early researchers such as Kitano [11] and Gruau [12] to propose evolutionary developmental systems (EDS), in which the genotype codes for a process through which the phenotype emerges. This parallels the way DNA instructions define the growth pattern of an organism. In most such systems, an important contributor to complexity is the context dependence of the interpretation of the genotype. The same genotype component may generate different phenotypic components depending on its history and context, just as the expression of a particular gene in an organism may depend on its cellular history and context.

A variety of representations have been used for EDS, many based either on Lindenmayer systems (L-systems [13]–[15]) or on cellular automata (CAs [16]), but with many other variants [17], [18].

EDS have been conspicuously successful, generating a wide variety of research, enough to form a separate track in recent GECCO conferences. However development is a complex process, and its interaction both with evolution and with the external environment is even more so. A wide range of systems have been built, incorporating different aspects of these interactions.

A. The Interaction between Evaluation and Development

We are specifically interested in how the developmental process interacts with evolutionary fitness evaluation. Many EDSs [11], [12], [14], [15], [19]–[22], that emphasise other aspects, have a relatively simple model for this. Individual genotypes are generated through normal evolutionary processes, the resulting genotype is then grown through its developmental processes, and the developed phenotype is evaluated through its environment. This is a reasonable abstraction, but it leaves out an obvious aspect of biological developmental systems, that the phenotype is not evaluated just once, but rather continuously throughout development: there is no point in having the genes required for an olympic athlete if one does not survive the embryonic stage.

A wide range of other EDS do evaluate the fitness of individuals during development. In some, the evaluation is of the same problem, repeatedly re-evaluated. Viswanathan and Pollack [23] demonstrated that evaluation during development could speed up evolution on a single problem, by using good solutions found at earlier stages of development. Miller and Banzhaf [24] guided their system to evolve and develop flags and Boolean circuits by repeatedly presenting the desired fitness function during the process of development. McPhee et al.'s IFD N-Gram GP system [25] goes one step further, using repeated evaluation of the fitness function not only to evaluate the fitness, but also to impose environmental feedback on the developmental process.

In biology the fitness function changes during the lifetime of an individual. In simpler cases, the change is random, or results from coevolution. This has commonly been simulated in artificial worlds such as Tierra [26].

Jung's [27] topological model for neural network development is repeatedly evaluated during the creature's development through predator-prey interactions. Kowaliw and Banzhaf's angiogenetic system [28] incorporates local evaluation of the circulatory system during the process of development. In a practical application, Tufte and Haddow [29], [30] investigated the structures and functionalities that could be achieved by FPGA-based developmental systems, evolved on a single problem but then generalised to other problems.

Although not usually viewed as an EDS, Stoffel and Spector's ontogenetic programming [31] also sees a varying fitness function during the course of its development/execution.

The idea of program execution as development has been taken much further in Harding et al.'s SMGGP [32]. Again, the program's self-modification throughout the course of development effectively changes the fitness landscape. Interestingly, SMGGP has been applied to Boolean parity problems [33] similar to those reported here, though differences in the function sets make it difficult to compare directly. However it is apparent that both developmental systems show substantially increased performance when compared to non-developmental systems.

In many animals, and most plants, that is all there is to it: the new organism is seeded into its environment, and has to grow through all stages in that environment. However it is noteworthy that across the various phyla of higher animals, a further behaviour has arisen multiple times in unrelated phyla: control of the environment by the parents, so as to provide the child with a staged series of environmental challenges, increasing in complexity as the child matures. Some hints that this might not be accidental can be seen in the gradual evolution of the complexity of this staging, from the simple protection of fish eggs by their parents in some species, through the intricate brooding mechanisms of many bird species, to the complex gestational and parental care mechanisms of mammals.

This staged exposure to more complex environments may play another role. Each stage of development must be well enough adapted to its corresponding environment to survive, but also be sufficiently flexible to generalise to the next environment at the next stage. An organism which overfits to the problem at one stage will find it difficult to adapt later, and so will be out-competed by individuals that, while solving the previous environment well enough, are adaptable to future environments. In accordance with parsimony theory from machine learning (see [34], ch. 7), this leads to preferential selection of more simply structured individuals, and also to control of runaway expansion in genotypic complexity.

B. Layered Learning

Incremental learning problems are well-known in machine learning; the earliest reference we know is in de Garis' work on evolving neural controllers for robot locomotion [35], but is better known through Stone and Veloso's work on layered learning [36]: a series of problems are presented to a learner, in an order which allows the solution of one to assist with the next.

Layered learning has a long history in non-developmental EC [37]–[39]. In these systems, problems are presented in order to the evolutionary system, and the population at the end

of each round of learning is used to seed the initial population for the next layer.

In EDS, layered learning has been used in the work of Bolouri et al. [40], in gradually building up the task complexity (in this case, neural visual processing) by evolving first for a simple task, then for incrementally more complex tasks. In these systems, the layered problems interact with the evolutionary system in the same way as in non-developmental systems; there is no direct interaction between the layered learning problems and the developmental process.

However this differs from our focus. In this work, we more directly analogise parental control of the developmental environment. That is, the layered learning stages are presented, not to the system as a whole, but to each individual during development, one layer per developmental stage. If the individual is uncompetitive at one layer/stage, it does not get to progress to later ones. We implement this through an incremental tournament selection mechanism that we detail in Subsection V-C.

Sekanina and Bidlo's system [41] is close to this in spirit, in emphasising the interplay of evolutionary and developmental dynamics. However they use an additive fitness function over the fitnesses at each stage of development, so that good fitness at a later stage in development can compensate for poor fitness at an earlier stage, thus reducing the evolutionary pressure to produce generalised individuals. A similar approach is taken by Krohn et al. [42] in using an additive fitness in a fractal protein developmental system to approximate the digits of π.

1) Are Biological Problems Really Layered?: It is important to clarify exactly what we are claiming about biology in pursuing this analogy. We do not claim that the complexity of biological developmental processes increases monotonically throughout development. Far from it. Angiogenesis, as studied by Kowaliw and Banzhaf [28], provides a clear example. Angiogenesis starts off slowly during development, then gradually speeds up – but reaches a peak sometime during childhood, then tails off. Kowaliw and Banzhaf's results suggest that the later stages may be governed by local search rather than genetic control, so that from the control perspective, the peak of complexity may come even earlier.

This is not in conflict with our claim, which is that the complexity of the fitness evaluation at these stages increases monotonically (or nearly so) – for respiration and circulation as for everything else, at least during the evolutionarily-relevant period up to reproductive maturity. Initially, there is no respiratory problem for the developing embryo – it can handle all respiration through diffusion. By day 22, the embryo has a complex beating heart. But it does not need it. Substantially larger animals than the 4mm embryo can survive perfectly well without one – even in the absence of a free supply of richly-oxygenated maternal blood. The embryo has a heart at day 22, not because it needs it then – it will not die then if it fails to beat – but because it will need it later and day 22 is an appropriate stage to develop it. This requirement is imposed, not by the day-22 fitness evaluation, but by fitness evaluations much later in the development process. The respiratory requirements themselves do change monotonically (and at birth, in a quite large step) throughout development.

C. Requirements for Evolutionary Developmental Evaluation

To fully model these aspects of biological evolution, a number of components are required:

1) *Developmental process governed by 'genes'*: The underlying genotypic representation should support both an evolutionary process (that is, it must be evolvable) and a developmental process (that is, it must incorporate a developmental process controlled by the genes).
2) *Developmental evaluation*: It should be feasible to evaluate the representation yielded by the developmental process at each stage of development.
3) *Layered learning*: The complexity of the problems handled should increase throughout development.
4) *Evaluation in sequence*: The evaluation should be biased toward earlier stages. The more difficult problems are evaluated sequentially during the developmental process.

In some of the experiments we describe below, we make use of two further abstractions from biology:

5) *Adaptive variation rates*: Good building blocks found at earlier evolutionary – and hence developmental – stages should be subject to reduced evolutionary pressure at later stages (leading to the effect sometimes referred to in biology as "ontogeny recapitulates phylogeny" – that is, embryological development to some extent parallels evolutionary development).
6) *Varying semantics during development*: Mechanisms are needed for genotypic elements to generate different phenotypic effects at different stages of development.

D. Implementations

We introduced a version of the above mechanisms in [43]; however that system involved a trivial development process. A version similar to that presented here, but with more rudimentary mechanisms for controlling the developmental process through meta structures, was presented in [44].

III. Regularity in Evolutionary Systems

We alluded earlier to a hypothesis that a staged interaction between evaluation and development may impose generalisation pressures on evolution, and thus can then lead to more regularly-structured individuals than would arise in non-developmental systems. We introduce here some of what is known about the growth in complexity and/or regularity in both artificial and biological evolutionary systems.

Individual complexity has been most heavily studied in genetic programming (GP), developed by Koza, Cramer and others [6], [45] with the intention of automatically solving problems using computers. Based on observations of biological systems, GP uses an abstraction of Darwin's natural selection mechanisms to evolve populations of solutions to problems.

A core problem in GP research is the phenomenon of bloat: GP generates solutions with large amounts of irregular and unnecessary code, that dramatically increases over time, and is not proportionate to any increase in the quality of solutions. Initially, the analogy was drawn with biological systems. For

example, the human genome uses less than 5% of its $3.2 * 10^9$ amino acid codons (ACTG) to encode proteins. At first, the remainder was thought to be largely non-functional [46], and in this way analogous to some forms of GP bloat.

However this analogy is doubtful on a number of grounds. It has become apparent since the original decoding of the human genome that the situation is more complex than originally thought. Substantial portions of the genome code for useful RNA, and are under active selection pressure [47], [48]. In addition, the growth of the biological genome does not resemble that of GP. In the $2 * 10^9$ years of eukaryotic evolution, the genome size has fluctuated widely, by up to five orders of magnitude, but there is no evidence whatever of any trend toward increasing growth [49]. There is currently only limited understanding of the drivers of biological genome size, or even whether it is under active selective pressure [ibid], but there can be no doubt that it bears little resemblance to code bloat in GP.

Bloat in GP results in irregularly-structured solutions, and this in turn leads to difficulties in scalability. Biological genotypes are also fairly irregular and not too compressible [50]. However when one comes to the higher-level effective code, the parts of the genome that are subject to selection, it is apparent that regularity and structure abound, as epitomised by the homeobox genes that control segment development in all bilateral animals [51], [52]. Their typical arrangement in the genome directly reflects their phenotypic effect on segments, with genes occurring along the chromosome in the same order as the segments whose phenotype they control [53]. In this regular structure, they differ very greatly from the genotypes that arise in GP.

A. Regularity and Modularity

The terms *Functional modularity* and *Structure regularity* were first introduced to evolutionary computation by De Jong [54]. Around the same time, Woodward [55] explained a module as "a function that is defined in terms of a primitive set or previously defined modules". Lipson subsequently [56], [57] defined functional modularity as the "structural localisation of function" and structure regularity as "the compressibility of the description of the structure" or "the correlation of patterns within an individual, such as symmetry, repetition and self-similarity, allowing evolution to specify increasingly extensive structures while maintaining short description lengths". Lipson argued that the terms modularity and regularity are often confused in the literature through the notion of re-use. Indeed, useful modularity can be repeated at higher stages of the development as good building blocks. Also, structural regularity appearing with a repetition of a pattern could be understood as functional modularity.

In Computer Science, "modularity" connotes encapsulation and re-use. In biology, it seems to be used more broadly, and is often applied to repetitions of patterns where no mechanism for re-use is proposed. For these reasons, the term can readily cause confusion. Since in this paper we are primarily interested in the repetition and variation of sub-structures (in GP terms, re-use of building blocks), we try to avoid such confusion by using the term 'regularity' wherever possible.

It has been argued (e.g. by Lehre [58]) that the emergence of regular structures requires a developmental process. It is possible that this emergence may be further enhanced by the generalisation pressures imposed by repeated evaluation during development. Testing this is an important focus of the experimental work reported here.

B. Regularity in Evolutionary Developmental Biology

How important and widespread is the development of regularity in evolutionary biology? Examination of nature shows that regularity and modularity have repeatedly evolved.

In this subsection, we first cover some background in Evolutionary Developmental Biology (Evo-Devo). We then discuss how regularity and modularity of structure arise in genomes.

1) Evolutionary Developmental Biology (Evo-Devo): Evolutionary developmental biology, often informally known by the term "Evo-Devo", is the study of the relationship between evolution and development. It is an old area of study, first investigated in the now somewhat overshadowed and discredited work of Haeckel [59]. It was however, resurrected and put on a sound scientific footing by Gould in "Ontogeny and Phylogeny", 1977 [60], with an emphasis on the importance of heterochrony (developmental change in time) "as a mechanism for evolutionary change" [61]. Lewis in 1978 proposed the general field of "Evo-Devo", stemming from the discovery of homeobox (Hox) genes, in particular the homeotic genes Ubx and abd-A and their role in inhibiting abdominal appendage formation in insects [62].

Biologists use the concept of Evo-Devo to understand morphological structures. Morphological changes in evolution generally result from developmental changes. Thus we need to understand developmental evolution in order to understand morphological evolution. For example, the diversity of *Ubx* binding sites (gain or loss) in the *cis*-regulatory region drives the diversification/distinction of butterfly hindwing patterns and of insect Hox-independent forewing patterns [63], see Figure III-B1.[1]

Evo-Devo opens the 'black box' to reveal the causes of the great variations in morphology of complex animals. Many genes controlling morphological development have now been identified, and the role of changes in these genes in driving phenotypic change between species is now being unravelled. As Carroll put it, "all complex animals – flies and fly catchers, dinosaurs and trilobites, butterflies and zebras and humans – all share a common 'toolkit' of 'master' genes that govern the formation and patterning of their bodies and body parts" [64].

Research in Evo-Devo has provided strong confirmation that regularity and modularity are essential for evolutionary development [53], [65]–[67].

2) Regularity and Structure in Biological Organisms: For simpler organisms, there is strong evidence of evolutionary pressure for compressed and regular genotypes (see, for example, [68]). In the current state of knowledge, evidence for gross regularity in the genome of higher organisms is harder to find. Despite perhaps 20 years of research, the best current

[1]Reproduced by kind permission of the author.

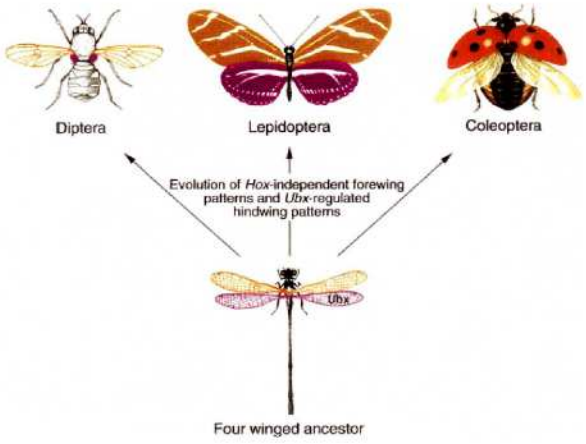


Fig. 1. Hox Genes and Divergence of Morphological Structures [63].[1]

compression achieved for whole eukaryotic genomes is a little over 1.5 bit per base [69], i.e. a compression ratio no better than 0.375. It may be that eukaryotic non-coding regions are not under selective pressure for regularity, or perhaps we have simply not yet identified the structure of this regularity. Nevertheless, there is indirect evidence that regularity is useful for eukaryotic organisms, and that mechanisms have arisen to promote it: the widespread emergence of duplication mechanisms. Repetition of regular structures in biological organisms has increased through repeated duplications on many scales. The mechanisms involved in these processes are quite different, suggesting that repetition may not be merely accidental, but selected for. We briefly review a few, to draw out how important these mechanisms are to evolution.

a) Genome Duplication: Whole or part genome duplication is common across the biological kingdoms. The yeast genus Saccharomyces, has experienced a whole genome duplication [70]. In plants, it occurs sufficiently often that new terms (tetraploidy, polyploidy) have been defined to describe it [71]. The entire animal genome has been duplicated at least twice prior to the split between tetrapods and fish [72] – and once again in the main fish lineage.

b) Gene Duplication: Ohno [73] argued that gene-level duplication is a key evolutionary driver. A duplicated gene sees reduced selective pressure, releasing it from some of mutation's disadvantages. Genes often fulfil multiple functions. Once a gene has been duplicated, both copies are free to specialise on some of the functions, without risk of loss of function.

Gu [74] found that yeasts with mutations in a single copy of a gene grew slower than did those with mutations occurring in one copy of a duplicated gene. By these mechanisms, gene duplication can establish sophisticated expression regulation. For example, the original green-sensitive opsin of the primate ancestors has split into two separate opsins with different (red and green) sensitivities in hominoids and old-world monkeys, resulting in their much improved color vision [75].

In *Drosophila melanogaster*, an ancestral gene, *janusA* had various roles, encoding two slightly different proteins in multiple tissues of both sexes, and in sperm (Figure 2). After more than 35 million years, a duplicate of *janusA* mutated to *janusB*, which specialised to encode a sperm-specific protein. During the next 15 million years, *janusB* generated another variant after duplication, *ocnus*, which specialised to encode another sperm-specific protein.

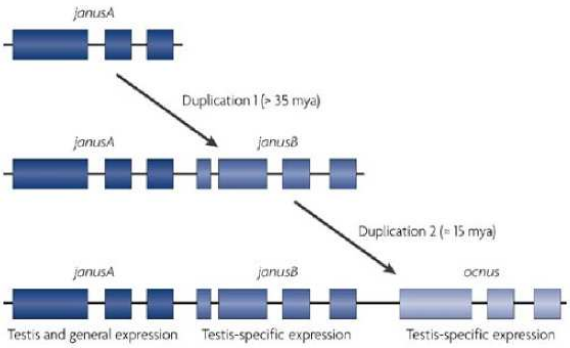


Fig. 2. An Example of Gene Duplication [76].[1]

c) Segmental duplication: Segmental duplication is derived from the repeated transpositions of small portions of chromosomes. Perhaps the best-known are the Alu elements, ~ 300 bp segments derived from the 7SL RNA gene, which appeared shortly after the evolution of the primates, ~ 65 mya [77]. While their overall function is unknown, there are around 1.4 million in the human genome, occupying $\sim 10\%$ of the total genome (greatly outweighing genes). New insertions occur around every 200 births. Another 20 kbp segment [78] of chromosome 16, LCR16a, has generated 15-30 copies in a 15 Mb section of the short arm of human and chimpanzee chromosome 16 (between 12 and 5 milion years ago).

C. Developmental Control

The biological developmental process is highly complex, incorporating both feedback within the developmental system itself, through complex control structures [79], and feedback from the environment. These feedback mechanisms are important, and finding suitable abstractions for them in EDS is a high priority [25], [28]. One important consequence of these control systems is that the developmental stage is implicitly available to the developmental process as one of its inputs. In this work, we provide a very crude abstraction through the 'meta' mechanisms we describe later.

IV. Technical Background

In this section, we introduce some of the tools that have been used to build a system meeting the requirements from section II. The main components are Lindenmayer systems (L-systems) and Tree Adjoining Grammars (TAGs).

A. D0L-Systems and Development

L-Systems were introduced by Lindenmayer in 1968 [13], using the central concept of a rewriting mechanism to simulate

the developmental processes of natural organisms. They are closely related to Chomsky grammars [80], the essential difference lying in the method of applying productions. In Chomsky grammars, productions are applied non-deterministically, whereas in simple L-systems they are applied in parallel, to 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. A deterministic and context-free L-system, also known as a Deterministic L-system with 0-interactions (D0L-system), is the simplest type of L-system. In these deterministic systems, exactly one production applies to any symbol of the L-system alphabet, and the productions are also context-free. The formal definitions describing a D0L-system and its operations are given below. A D0L-system [13] is an ordered triplet $G = (V, \omega, P)$ where:

- V is the alphabet of the system, V^* the set of all words over V.
- $\omega \in V^*$ is a nonempty word called the *axiom*.
- $P \subset V \times V^*$ is a finite *set of productions*. A production $(p, s) \in P$ is written $p \to s$; p and s are the *predecessor* and *successor* of this production.
- Whenever there is no explicit mapping for a symbol p, the identity mapping $p \to p$ is assumed.
- There is at most one production rule for each symbol $p \in V$.

Let $p = p_1 p_2 ... p_m$ be an arbitrary word over V. The word $s = s_1 s_2 ... s_m \in V^*$ is directly derived from p, denoted $p \Rightarrow s$, iff $p_i \to s_i$ for all $i \in \{1 \dots m\}$. If there is a *developmental sequence* $p_0, p_1, \dots, p_n$ with $p_0 = \omega$, $p_n = s$, and $p_0 \Rightarrow p_1 \dots \Rightarrow p_n$, we say that G generates s in a derivation of length n.

In short, D0L-systems operate on sequences of symbols called *strings* or *words*. In a single *derivation step*, each letter in the predecessor string is replaced by its successor using the applicable production from the production set P. The *developmental process* is simulated as a sequence of such derivation steps, beginning with a given initial string, called the *axiom*, and denoted ω.

An example of a D0L-system is shown as in Figure 3 at three stages of development. The developmental process is generated by the D0L-system $G = (V, \omega, P)$ with alphabet $V = (S, A, B, C, x, y, z)$, axiom $\omega = S$, and the production set P given by:

- $P_1 : S \to xAB$
- $P_2 : A \to yBzB$
- $P_3 : B \to xCzx$
- $P_4 : C \to A$

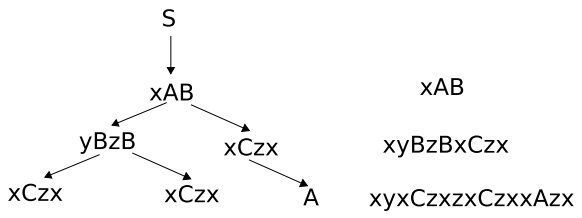


Fig. 3. An Example of D0L-system

B. Tree Adjoining Grammars and TAG3P

Tree Adjoining Grammars were introduced by Joshi [81] to overcome some perceived problems with Context Free Grammars (CFGs) for representing natural language. They are based on the insight, that a sentence such as 'The big black cat sat licking its paws on the plush, comfortable mat, which it had commandeered' may be analysed as a basic sentence (a so-called α-tree) 'The cat sat on the mat', into which insertable elements (β-trees – 'big', 'black', 'licking its paws', 'plush', 'comfortable', 'which it had commandeered') have been inserted (adjoined) at grammatically-appropriate places – and that these basic and insertable elements can themselves be further decomposed in the same way. TAG representation then consists of a tree recording the insertions and substitutions of new components into the so-far-constructed sentence. TAG grammars can just as readily be built for arithmetic or other expressions as for sentences (TAG grammars subsume CFGs: every CFG can be represented as a TAG). The representation has many advantages, both for natural language processing and for GP, but in this context the key advantage is one of completeness: any rooted subtree of a TAG tree represents a complete individual which can be immediately evaluated ('The cat sat on the mat', 'The black cat sat on the mat', 'The big, black cat sat on the mat', ...). In this work, for technical reasons, we use lexicalised TAGs (TAGs in which every elementary tree has at least one lexical element on its frontier).

The use of TAG representation to encode expressions, and more specifically as a GP representation, is described in detail in Nguyen et al. 2006 [3]. For convenience, we reprise from that paper the formal definition of a tree adjoining grammar:

A tree adjoining grammar is a tree-rewriting system consisting of a quintuple $T = (\sum, N, I, A, S)$, where:

1) $\sum$ is a finite set of terminal symbols.

2) N is a finite set of non-terminal symbols: $N \cap \sum = \emptyset$.

3) S is a distinguished non-terminal symbol: $S \in N$.

4) I is a finite set of finite trees, called initial trees (or α-trees).
In an initial tree, all interior nodes are 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 $\downarrow$ (for substitution).

5) A is a finite set of finite trees, called auxiliary trees (or β-trees).
In an auxiliary tree, all internal nodes are labeled by non-terminal symbols, and a node on the frontier is labeled either by a terminal or non-terminal symbol. The frontier must contain a distinguished and unique node, the foot node, labeled by the same non-terminal symbol as the tree's root node, and marked with an asterisk (*); other nodes on the frontier labeled by non-terminal symbols

are marked with $\downarrow$ (for substitution).

The trees in $E = I \cup A$ are called elementary trees. Initial trees and auxiliary trees are denoted α and β respectively. A tree with its root labeled by a non-terminal symbol X is called an X-type elementary tree.

In essence, an α-tree with all terminal symbols on its frontier is just like a minimal complete sentence, while a β-tree is a minimal recursive structure used to modify complete sentences.

Tree Adjoining Grammar Guided Genetic Programming (TAG3P) – the system on which DTAG3P is based – is a typical grammar guided GP system, with the sole exception of the use of TAG derivation trees, rather than CFG derivation trees, as the individual program evolutionary representation [3].

C. Measuring regularity in Evolutionary Computation

Much EC research has aimed to generate modular, regular solutions [14], [44], [82]–[85]. However developing a metric to measure regularity and modularity, and hence to objectively compare the effectiveness of different mechanisms in promoting regularity and modularity, has only rarely been attempted. Hornby recently developed such metrics [86], [87], but assumed an explicit and specific representation of modularity (the computer science perspective), which is poorly suited to measuring emergent regularity (the biological view). We use an alternative based on compression.

Individuals with structural regularity have a repeated substructure or pattern; hence they should be more compressible. Thus measuring compressibility provides one way to implicitly measure specific kinds of modularity (those that correspond to the coder's assumptions) [88].

Data compression is the process of encoding data using fewer bits than the unencoded raw data. Compression algorithms include two components: the *model* and the *coder*. The model captures the probability distribution of the original data by discovering regularities in its structure. The coder takes advantage of the resulting probability biases to generate efficient coding of the same data.

Our data consists of GP trees generated from a variety of GP systems. Thus we use a tree compression algorithm, XMLPPM [89], which is an extension of the Predict by Partial Match (PPM) model [90] to the compression of trees.

In GP, code bloat is almost inevitable [6], [91]–[93]. In Figure 4 we see an example (a), which may be replaced by a smaller tree (b) with identical evaluation. Ineffective code may incorporate a large amount of repeated code, hence it may change the compression ratio of the tree. Thus any regularity metrics should measure the compressibility, not only of the whole solution genotype tree, but also of the effective part. This requires reliable methods of eliminating ineffective code.

To find the effective code, we use 'equivalent decision simplification' (EDS) [88] to convert a tree into an equivalent smaller tree. EDS determines the equivalence of code segments by semantic checking, testing whether they are equivalent over a set of fitness cases, in place of the (necessarily incomplete) pre-determined sets of rules used in syntactic simplification [94]–[96]. This semantics-based approach has been shown [88] to generate substantially greater simplification than the more commonly-used syntactic simplification.

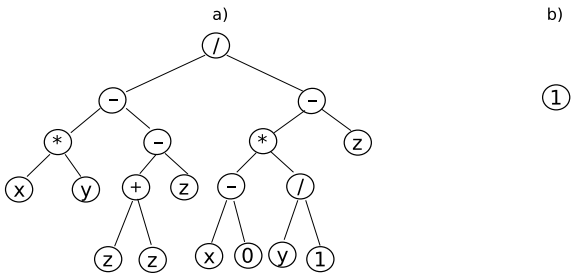


Fig. 4. GP Code Bloat Example (a) and its Simplification (b)

V. The DTAG3P System

DTAG3P is the core system in this work. It is built on the earlier non-developmental TAG3P system, in which TAG grammar trees act as genotypes, and are transformed in turn into CFG trees and then expression trees for evaluation. This seemingly complex transformation is used for good reason: TAG trees have important properties absent from CFG and expression trees. In this context, the most important is feasibility. CFG and expression trees are difficult to use in developmental systems, because it is difficult to extend them while retaining validity (which probably explains why few, if any, developmental systems use classic tree-based GP representations). TAG trees don't have this problem. Any legal extension of a valid TAG tree, will generate a new valid TAG tree. Thus they form an ideal substrate for a developmental tree-based GP. However if TAG trees are to form the developmental framework, we still need a genotype representation. We chose to modify D0L L-systems so that they could support the growth of TAG trees. Thus the genotype consists of a set of D0TL rules, a tree-based analogue of D0L rules. These are expanded during development to grow a TAG tree; at each stage, the TAG tree is transformed into a CFG and then into an expression tree (as in TAG3P) to permit evaluation. Figure 5 shows an outline of the structure of the overall system.

A. Genotype encoding, TAG-based L-systems

The D0TL system takes on the role of DNA, being the subject of the evolutionary variation operators, but it is not directly evaluated. The D0TL system encodes the instructions for growing a TAG derivation tree (somewhat in the way DNA encodes instructions for growing proteins), but that is not the direct subject of evaluation either. Rather, just as the growth of proteins and their cellular organisation creates the organism (the phenotype) that is actually evaluated, so our TAG derivation trees then generate first CFG derived trees, and then GP expression trees, which are directly evaluated, and hence are described as the phenotype. For want of a better term, we describe the TAG and CFG trees as "intermediate phenotypes".

Deterministic Tree L-systems with 0-interactions (D0TL-systems) are a generalisation of D0L-systems, in which the Right Hand Side (RHS) of a rule, instead of being restricted

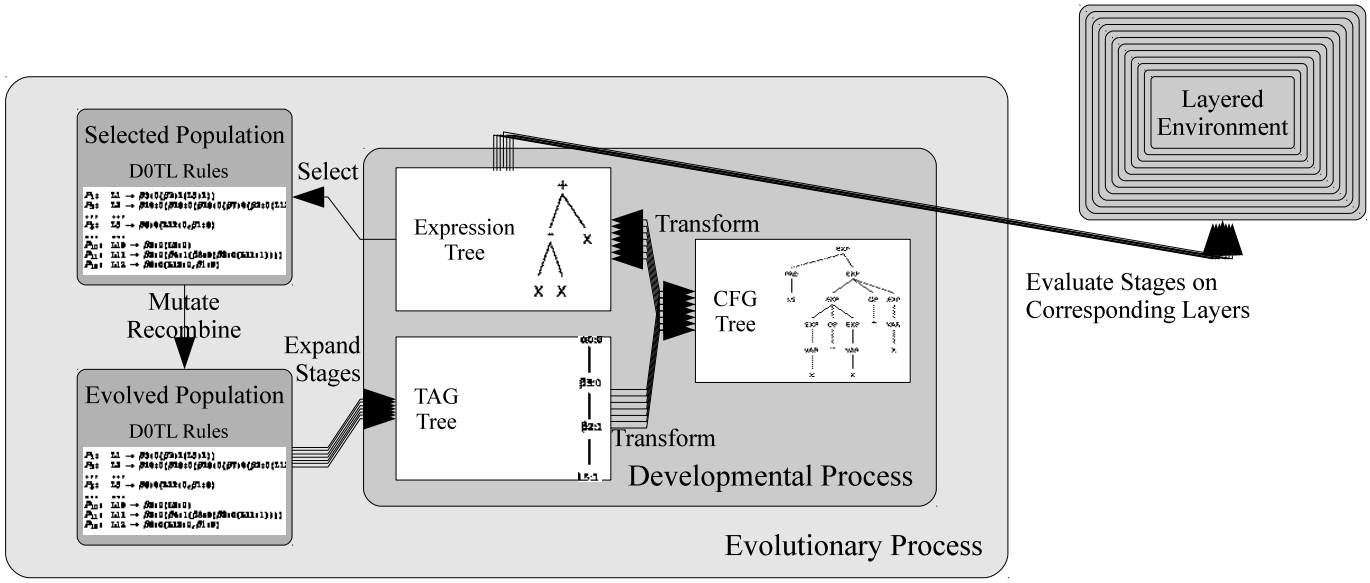


Fig. 5. Schematic Structure of DTAG3P Evolutionary / Developmental / Layered Learning System

to a string as in a D0L-system, is a tree. It nevertheless contains a mixture of non-predecessor and predecessor nodes.[2] In the particular variant used here, predecessor nodes are only permitted to occur on the tree frontier of β trees. An example of a D0TL-system appears in detail in the next Section.

We assume there is a pre-defined TAG grammar $T = (\sum, N, I, A, S)$, together with a new set of (L-system) nonterminals $L = \{L_1, L_2, L_3, ...\}$. A D0TL-system in this representation comprises a triple G (V, ω, P), where:

- $V = L \cup A$: that is, the alphabet, consists of the set $L = \{L_1, L_2, L_3, ...\}$, the L-system nonterminals (i.e. symbols that can act as the left-hand-sides of L-system rules) together with the set A of auxiliary (β) trees from T, which act as the terminals of the L-system grammar.
- The initial axiom ω consists of an element of I (an α tree), in which one of the adjunction locations has been marked by one of the L_i.
- The set of rewrite rules $P = \{P_i : i = 1...m\}$ have the form $P_i : L_i \rightarrow \mathcal{T}(S_1, S_2, ..., S_n)$ where the right-hand side is an (extended) auxiliary (beta) tree in TAG, with each S_i being an element of V, and $\mathcal{T}$ being a tree built from them. Any nonterminals (L_j) must lie on the frontier of the tree.

As an example, we might have a TAG grammar $T' = (\sum', N', \alpha_1, \beta_1, \ldots, \beta_8, S')$ and a D0TL system G'=(V', ω', P') with V'=$\{L_1, \ldots, L_{12}\}$, ω' = $(\alpha_1 L_1)$, and P' containing productions $P_1 \ldots P_{12}$ with, for example, productions P_1, P_2, P_5, P_{12} having the right hand sides shown as L_1, L_2, L_5, L_{12} in Figure 6. We will also use a bracketed notation as shown in Table I. The bracketed notation has the following meaning: $\beta_i(\beta_j : l_j, \beta_k : l_k)$ is to be interpreted as the adjunction of auxiliary tree β_j at location l_j in parent tree β_i, and of auxiliary tree β_k at location l_k. The locations are numbered in this way: the root is labelled as 0, and then the only other adjunction location is labelled as 1.[3]

TABLE I
(PART OF) AN EXAMPLE D0TL-SYSTEM

P_1:	$L_1 \rightarrow \beta_3$:0(β_2:1(L_5:1))
P_2:	$L_2 \rightarrow \beta_{10}$:0($\beta_{10}$:0($\beta_{10}$:0($\beta_7$:0($\beta_2$:0($L_{12}$:1)))))
...	...
P_5:	$L_5 \rightarrow \beta_6$:0(L_{12}:0,β_1:0)
...	...
P_{12}:	$L_{12} \rightarrow \beta_6$:0($L_{12}$:0,$\beta_1$:0)

Figure 6 depicts tree representations of these rules, and shows the expansion of the initial tree through them. It starts with the initial tree in ω, α, and a corresponding predecessor P_1. P_1 can be replaced by the corresponding right hand side, L_1, resulting in the stage 1 individual. This tree now contains a predecessor P_5 (in general, we might have more than one, depending on the rules we evolved), which can then be expanded by L_5 to give the stage 2 individual. This process can be repeated until all developmental stages (here, 3) have been completed.

B. DTAG3P

DTAG3P uses the TAG-based D0TL system to encode the construction of TAG derivation trees, thus defining the language bias of the genetic programming system. DTAG3P proceeds through evolution of the L-system rules. In the current version, the DTAG3P system maintains a fixed size set of rules, though there is no in-principle difficulty in making the

[2]Because the predessor nodes behave at some times like nonterminals, and at others more like terminals, we use the terminology predecessor / non-predecessor instead.

[3]In general, in TAG trees, we can permit adjunction to both the root and the foot of a β tree. However this can cause problems, because the same node can be both a foot and (after adjunction) a foot. There are difficulties in interpretation (how should we interpret such a node) and bias (root/foot nodes would be twice as likely to be adjoined as other nodes). For these reasons, we don't permit foot adjunction; since the highest arity our grammars use is binary, we only ever have at most one non-root adjunction location.

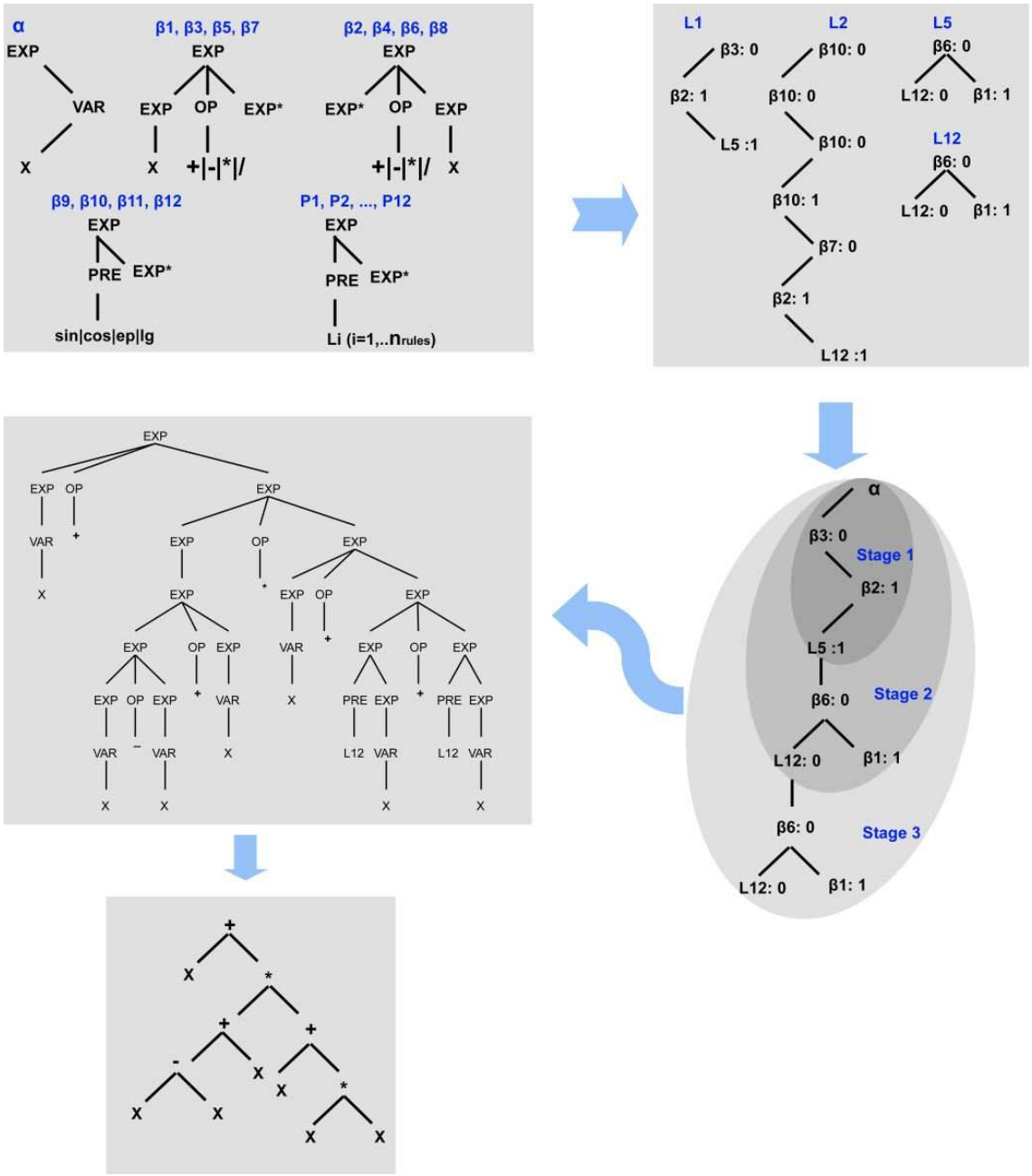


Fig. 6. An Example TAG-Based L-system and its Development through Expansion:
Top Left, pre-specified TAG grammar; Top Right, D0TL Rules;
Mid Right, First Three Stages of TAG Tree Development; Mid Left, Corresponding Third Stage CFG Tree;
Bottom Left, Corresponding Expression Tree

rule-set size variable (however variable rule-sets would require additional operators, and thus additional parameter setting).

A standard GP system [6], may be described using Koza's five-component scheme. We adapt it by adding a component to describe development, and by extending the evaluation description to incorporate developmental evaluation. Thus our specification consists of six components: representation, initialisation, development process, fitness evaluation, genetic variation operators, and parameters.

1) Representation: DTAG3P uses the TAG-based D0TL derivation trees as the individual genotype representation. The user specifies a TAG lexical grammar G_{lex}, defining the solution space for the specific family of problems.

The genotype-to-phenotype transformation uses the D0TL developmental process, generating a TAG derivation tree at each stage of development. This representation satisfies requirement 2 from Section II because of the 'feasibility' property of TAG derivation trees, in that a tree legally grown from an α tree is always semantically meaningful – it does not have to be 'completed' in the same way as a CFG tree. However, it still cannot be directly evaluated. As in the genotype-phenotype mapping used in TAG3P [3], the TAG derivation tree is used to create a second 'intermediate phenotype', a Context-Free Grammar (CFG) tree, which is then transformed

into an expression tree phenotype. As in standard GP, this expression tree is directly evaluated for fitness. The genotype-to-phenotype transformation can be summarised in Figure 7.

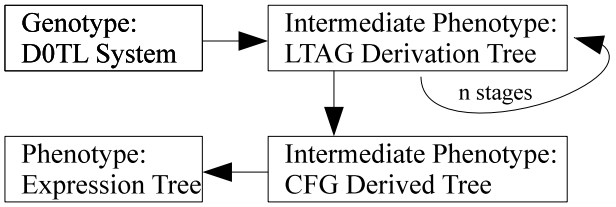


Fig. 7. DTAG3P Genotype-to-Phenotype Mapping Process

2) Initialisation Procedure: This is an algorithm for creating an initial random individual; it is repeated until the population of size max_{pop} has been created. Each individual is a D0TL system, containing n_{rules} rules $P = \{P_1, P_2, \cdots, P_{n_{rules}}\}$. We denote the predecessors of these rules as $V = \{L_1, L_2, \cdots, L_{n_{rules}}\}$. For each rule, we randomly select $\omega = (\alpha L)$: $\alpha \in A$, A being the set of initial trees in G_{lex} (TAG), and L $\in V$. We construct the successor (RHS) S_i of each rule by first randomly drawing β-trees from B, the set of auxiliary trees in G_{lex} (TAG) and assigning them, together with random adjunction locations, to the RHS of P_i, up to a random limit between $min_{betas}, \cdots, max_{betas}$; we then randomly draw n_{letter}, predecessors from V, and insert them into the RHS, at random adjunction addresses from the frontier of the tree.

TAG derivation trees are produced by decoding the D0TL-system. A parameter max_{life} is used to specify the number of cycles of replacement of letters by their successors (i.e., the number of developmental phases).

We can describe this initialization procedure through the pseudocode in Algorithm(1).

3) Development Process: The development process is as in Figure 6, a multi-stage expansion of the D0TL system, each stage being evaluated for fitness as described below.

4) Staged Fitness Evaluation: Each problem domain is represented, not by a single problem as in typical GP systems, but by a family of problems of increasing difficulty. At the first stage of development, the individual is evaluated against the simplest problem; at the second stage against the second problem, and so on.

In more detail, the process expands the D0TL system to a given stage, interpreting the result as a TAG derivation tree, and converting it successively to a CFG parse tree, and an expression tree. The latter is evaluated against the corresponding problem exactly as in a typical GP system. Each individual undergoes a fixed maximum number max_{life} of developmental stages (corresponding to the size of the problem family). Note that the fitness evaluations of later stages of an individual's lifetime might not be used in selection. Lazy evaluation would eliminate them from the computational cost. For analysis purposes in the subsequent experimental section, we perform the evaluations, but also report the computational cost had we avoided them.

Algorithm 1 Pseudocode for Initialisation Procedure

for $i = 1 \ldots max_{pop}$ **do**
  Randomly choose an α-tree α_i
  Set axiom ω equal to random predecessor L_j in α_i
  for $j = 1 \ldots n_{rules}$ **do**
    Set the default alteration rate p_{adapt} for the rule
    Select a random predecessor for $rule_j = L_j$
  end for
  for $j = 1 \ldots n_{rules}$ **do**
    Choose a random size $l = 1 \ldots max_{betas}$
    Pick a β-tree at random (β_t) and set $T = \beta_t$
    for $k = 0 \ldots l-1$ **do**
      Uniformly random pick a node *n* $\in$ T with
      at least one unused adjoining address
      Randomly pick an empty address *a* in *n*
      Choose tree t from the β-trees in G_{lex}
      that can adjoin to *a*
      Adjoin t to *a* in T
    end for
    for $m = 0 \ldots n_{letter}$ **do**
      Adjoin random $L_p \in V$ to a leaf location in T
      Set successor of $rule_j$=T.
    end for
  end for
end for

C. Selection Mechanisms

Tournament selection is used because it provides a mechanism to handle a family of problems, such as DTAG3P uses. Individuals are developed to stage 1, and evaluated on the first problem. All 'equal best' individuals are retained for stage 2, the others being eliminated. The individuals are then developed to stage 2, and are evaluated on the second problem; again, 'equal best' individuals are retained for stage 3, the rest being eliminated. The process continues until only one individual remains: it is then selected as the result of the tournament. If all possible stages have been evaluated (i.e. some individuals are always 'equal best' for each stage), then one of them is chosen uniformly randomly.

'Equal best' requires some explanation. To avoid deciding tournaments by minute differences, when a later stage might be able to make a more rational determination, we allow some 'slop' in equality. A tolerance value δ is one of the parameters of the algorithm; individuals which differ from the true best by less than δ are retained for the next stage.

Algorithm 2 Pseudocode for Selection Mechanism

$i \leftarrow 1$
while $|fit(I_1, i) - fit(I_2, i)| < \delta$ **do**
  $i \leftarrow i + 1$
end while
if $fit(I_1, i) < fit(I_2, i)$ **then**
  I_1 wins
else
  I_2 wins
end if

Denoting the the fitness of individual I evaluated at stage s by $fit(I_s)$, for two individuals (I_1, I_2), the comparison process (for minimisation) can be formalised as the pseudocode in Algorithm 2. An example of this algorithm is shown in Figure 8, comparing the individuals I_1 and I_2 with fitness value arrays (corresponding to the 9 different stages), I_1(10.05, 14.66,. . .,20.35), and I_2 (10.06, 14.66,. . ., 10.35). In this case, I_2 would be chosen for further evolution.

D. Genetic Operators

This subsection discusses the main genetic operators in DTAG3P – the recombination operator, and three mutation operators: internal crossover, sub-tree mutation and lexical mutation.

1) Recombination: Recombination chooses two individuals, p_1 and p_2 from the population by the selection mechanism. It then uses them as parents to create two child individuals c_1 and c_2. Suppose parent 1 has the following rules:

$$P_{11} : \mathrm{L}_1 \rightarrow \beta_3 : 0(\beta_2 : 1(L_5 : 1))$$
$$P_{12} : \mathrm{L}_2 \rightarrow \beta_{10} : 0(\beta_{10} : 0(\beta_{10} : 0(\beta_7 : 0(\beta_2 : 0(L_{12} : 1)))))$$

and parent 2 has these:

$$P_{21} : \mathrm{L}_1 \rightarrow \beta_4 : 0(\beta_2 : 1(\beta_1 : 1(\beta_1 : 0((L_{12} : 1)))))$$
$$P_{22} : \mathrm{L}_2 \rightarrow \beta_7 : 0(\beta_9 : 1(L_7 : 1))$$

We provide the two operators of *rule exchange* and *subtree crossover*. The resulting effects on child 1 are shown below (boxes show the parts of child 1 that differ from parent 1).

a) Rule exchange: (somewhat analogous to chromosome-level exchange in biology) replaces one entire rule body with the corresponding one from parent 2:

$$P_{31} : \mathrm{L}_1 \rightarrow \beta_3 : 0(\beta_2 : 1(L_5 : 1))$$
$$P_{32} : \mathrm{L}_2 \rightarrow \boxed{\beta_7 : 0(\beta_9 : 1(L_7 : 1))}$$

b) Sub-tree crossover: (somewhat analogous to gene-level exchange in biology) randomly selects a sub-tree in a rule from parent 1, and a sub-tree with the same root from a rule in parent 2, and exchanges the sub-trees. This follows the general framework of sub-tree crossover in TAG3P [3].

$$P_{31} : \mathrm{L}_1 \rightarrow \beta_3 : 0(\beta_2 : 1\boxed{(\beta_1 : 1(\beta_1 : 0((L_{12} : 1)))}\,)$$
$$P_{32} : \mathrm{L}_2 \rightarrow \beta_{10} : 0(\beta_{10} : 0(\beta_{10} : 0(\beta_7 : 0(\beta_2 : 0(L_{12} : 1)))))$$

2) Mutation: Mutation chooses a parent p_0 using the selection mechanism, and creates a child *c* through applying one of the mutation operators defined below. We provide two mutation mechanisms causing change on varying scales: rule interchange and sub-tree mutation. Given parent 1 as before, we describe these in turn.

a) Rule interchange: (somewhat analogous to chomosome-level mutations such as duplication, deletion, fusion in biology) randomly interchanges the RHS of rules in the parents (it is thus a very large scale operator – a macromutation):

$$P_{31} : \mathrm{L}_1 \rightarrow \boxed{\beta_{10} : 0(\beta_{10} : 0(\beta_{10} : 0(\beta_7 : 0(\beta_2 : 0(L_{12} : 1)))))}$$
$$P_{32} : \mathrm{L}_2 \rightarrow \boxed{\beta_3 : 0(\beta_2 : 1(L_5 : 1))}$$

b) Sub-tree mutation: (somewhat analogous to gene-level mutations in biology) deletes a random sub-tree in a random rule (e.g. $\beta_2 : 1(L_5 : 1)$) replacing it with a newly generated sub-tree (e.g. $\beta_{11} : 0(\beta_8 : 0(\beta_9 : 1(L_1 : 0)))$):

$$P_{31} : \mathrm{L}_1 \rightarrow \beta_3 : 0(\boxed{\beta_{11} : 0(\beta_8 : 0(\beta_9 : 1(L_1 : 0)))}\,)$$

We note that this set of operators is fairly complete (for fixed-size rule sets), able to adapt the rule sets on a variety of scales. They also have some level of surface plausibility in analogy to biological systems.

3) Parameters: As with most evolutionary systems, it is necessary to specify a set of parameters to define the exact configuration of DTAG3P. They are shown in Table II. One parameter needs some detailed explanation. In biology, it is clear that early developmental processes can become fixed, and subject to lower rates of mutation than later processes (so that human embryos develop – and then reabsorb – gill-like structures despite some 350 million years since our most recent ancestors actually used gills). To simulate this in our system, when a rule is used in a developmental stage which was used to select the parent (but is not the final such stage), it is reset to a lower value of adaptation p_{good}. Thus the child is more likely to inherit this rule unchanged.[4]

- The number of generations per stage:	max_{gen}
- Population size:	max_{pop}
- The adaption rate:	p_{adapt}
- The reduced adaption rate:	p_{good}
- Rule exchange rate;	p_{RX}
- Sub-tree crossover rate:	p_X
- Rule interchange rate:	p_{RI}
- Sub-tree mutation rate:	p_{sub}
- Lexical mutation rate:	p_{lex}
- Reproduction rate	p_{copy}

TABLE II
DTAG3P EVOLUTIONARY PARAMETERS

For a developmental system, we also require further parameters to describe the developmental process. For DTAG3P, they are shown in Table III.

- The number of stages:	max_{life}
- The number of rules:	n_{rules}
- The minimum number of β-trees in a rule:	min_β
- The maximum number of β-trees in a rule:	max_β
- The number of predecessors in a rule RHS:	n_{pred}
- The minimum difference in each stage:	δ

TABLE III
DTAG3P DEVELOPMENTAL PARAMETERS

E. Meta Mechanisms

Previously (in Subsection II-C) we detailed a set of requirements that our developmental evaluation system needs to fulfil. As described so far, DTAG3P meets all but the last: it has no mechanism by which the specific stage of development can

[4]The application rates of operators sum to 1.0, so we don't directly specify p_{copy}, but derive it from $p_{copy} = 1.0 - (p_{RX} + p_X + p_{RI} + p_{sub})$.

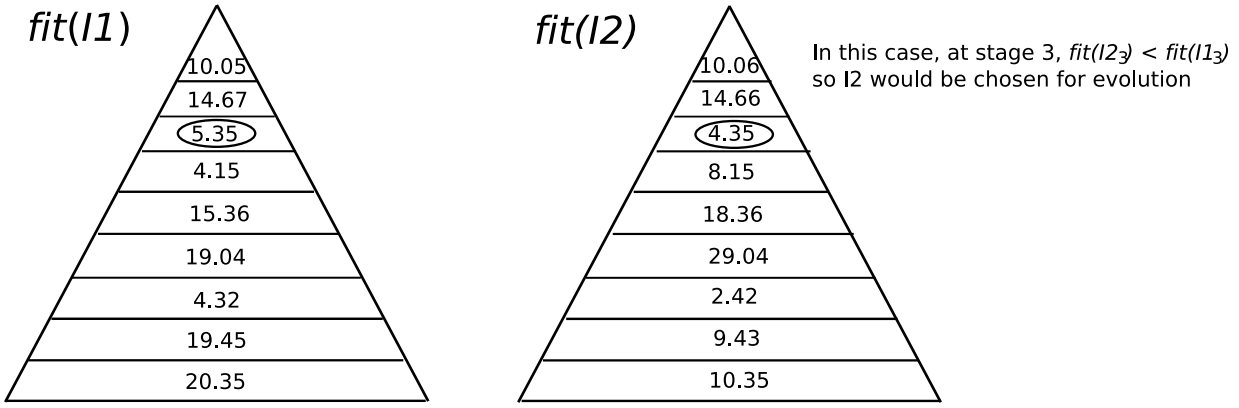


Fig. 8. Example of Selection through Developmental Evaluation

feed back into the process. Such a mechanism is desirable for two reasons:

1) Biological plausibility – almost all biological developmental systems do incorporate such a mechanism; without such a mechanism, we probably would not recognise a process (for example, aggregation of unicellular organisms into a colony) as truly developmental.
2) Practical necessity – most natural layered-learning problems naturally provide – or in many cases require – such a mechanism. For example, if the system has to learn a polynomial $\sum_{i=1}^{s} x^s$, at stage s, it seems only fair to give the system knowledge of the value of s. But even more critical, if the system is required to learn parity of size s at stage s, it should have this information to ensure that it does not generate expressions containing variables $X_t, t > s$ that cannot be evaluated at stage s.

Missing variables may be handled in other ways. In [97], we used $undef$ to deal with 'undefined' variables during evaluation. However this imposes a huge burden on the developmental system, because in the early stages of learning, almost all variables are undefined, so that it is very difficult during initialisation to generate individuals that have a defined fitness. While complex penalty mechanisms could be introduced, meta-variables seem to provide a more intellectually satisfying solution.

In this system, we provide two such mechanisms: meta-constants and meta-variables.

a) Meta-constants (MC): are treated somewhat like ephemeral random constants (ERCs) in standard GP. First, an MC C is sampled from a distribution just as an ordinary ERC, but for an MC, the distribution is limited to the range $-max_{life} \ldots \max_{life} -1$. To ensure that the actual value, when sampled at stage s is within the range $1 \ldots s$, the value is computed as $(C \mod s) + 1$ when it is being evaluated at stage s (when development proceeds to stage $s+1$, MCs that were evaluated at previous stages retain their previous values).

b) Meta-variables (MV): are analogous, but the value of an MV is a variable, rather than a constant. Given an MV V, its corresponding index I_V is evaluated at evolution time as for MCs – that is, its value is sampled from a distribution over $-max_{life} \ldots \max_{life} -1$. As with MCs, when it is being evaluated at stage s, it will produce the corresponding variable $X_{(I_V \mod s)+1}$; once an MV has been evaluated at a particular stage, it retains that value in subsequent stages, rather than being re-evaluated at a later stage.

VI. Experiments

In this section, we first describe the problem families that we used in our layered learning. We then describe the experimental settings we used to compare DTAG3P with TAG3P and standard (Koza-style) GP. Finally, we describe some variants which were used to test the importance of different components of DTAG3P.

A. Problem Domains

A number of problem domains have been used for testing in this work: symbolic regression problems, boolean *n*-parity problems and ORDERTREE problems.

1) Symbolic Regression of a Polynomial: In symbolic regression, the system is given a set of points to fit (as in linear and other kinds of regression – polynomial, logistic etc.), and the system is free to construct any functional form it chooses from its basic stock of functions. In this problem, the function and terminal sets consisted of $F = \{+, -, \times, /, sin, cos, ep, lg\}$ and $T = \{x\}$. The target polynomial symbolic regression family was the family:

$$\begin{aligned} F_1 &= x \\ F_2 &= x^2 + x \\ F_3 &= x^3 + x^2 + x \\ &\ldots \\ F_9 &= x^9 + x^8 + \ldots + x^3 + x^2 + x \end{aligned}$$

with the general form and recurrence relation

$$F_s(x) = \sum_{i=1}^{s} x^i \quad (1)$$

$$F_{i+1}(x) = x \cdot (1 + F_i(x)) \quad (2)$$

In these examples, the data consisted of 20 points sampled uniformly randomly from the interval (-1,1).

In typical experiments, GP scales up to F_4 or F_5; to understand the scaling of DTAG3P, the experiments were

continued to F_9. This problem family was chosen because it has been widely studied, and is especially well-suited to layered learning. We note that the problem family satisfies the recurrence relation 2, so that each problem constitutes a substantial building block for the next, and the progression from each stage to the next requires exactly the same mechanism.

2) Symbolic Regression of a Trigonometric Expression: The second target family was a little tougher, requiring the system to find a changing trigonometric relationship. A trigonometric symbolic regression problem family was chosen for similar reasons:

$$\begin{aligned} \Phi_1 &= sin(x) \\ \Phi_2 &= sin(2 \cdot x) \\ \Phi_3 &= sin(4 \cdot x) \\ &\cdots \\ \Phi_9 &= sin(2^8 \cdot x) \end{aligned}$$

with the general form

$$\Phi_s(x) = sin(2^{s-1} \cdot x) \quad (3)$$

This problem family also used 20 sample points, but sampled from the range $(-\pi, \pi)$.

3) Boolean Parity: The k-parity problems constitute a long-studied family of difficult GP benchmarks. The even (odd) task is to evolve a function returning 1 if an even (odd) number of the inputs evaluate to 1, and 0 otherwise. Langdon and Poli observed in [98] that the task is extremely sensitive to change in the function set, and that the commonly-used set $OR, AND, NOR, NAND$ omits the useful XOR and EQ building blocks. Inspired by Poli and Page [99], we chose the function set AND, OR, XOR, NOT as a suitable compromise – containing the XOR building block, unlike the first function set, but tougher than Poli and Page's set (which contained all binary Boolean functions). The problem family seems particularly well-suited to investigating scalability, since standard GP scales up to k=8, but not beyond.

4) ORDERTREE: The ORDERTREE problem, first introduced in Hoang et al. [100], was designed with an awareness of Daida's problem of structural difficulty [101], and so it attempts to remove the shape bias in optimal solutions. Hoang et al. experimentally verified that the difficulty of the ORDERTREE problem can be tuned both by increasing the size of the problem, and by increasing the non-linearity in the fitness structure. In essence, the ORDERTREE problem is a natural analogue of the ONEMAX problem [102], a popular genetic algorithm test problem. The function set and terminal set for ORDERTREE of size *n* are defined as: F(T)= {'1', '2', ..., 'n'}. Both the function nodes and the terminal nodes are also labeled with numbers from the set {1,2,...,n}. Note that all the functions are of arity 2 (i.e, each function has two arguments). The (maximising) fitness evaluation is based on a 'left-neutral-walk' procedure: 'If the value of a node is equal to that of its parent node, the fitness calculation continues by visiting the left child. If that new node's value is less than its parent, the process terminates [...], and no fitness contribution results, the whole subtree being treated as an intron. If the node value is greater, the subtree is evaluated, and the fitness contribution is passed to the parent. If the value is equal to its parent, its left child is evaluated recursively. In all cases, the fitness contribution of the right child is zero, so that the right subtree acts as an intron. The process is fully detailed in [100]. Figure 9 shows examples of fitness calculations for a 3-ORDERTREE problem (top) and one of many optimal solutions (bottom).

The ORDERTREE problem family used in this paper was $O(1), O(2), \ldots, O(6)$, where $O(i)$ denotes the ORDERTREE problem of size i.

B. Meta Mechanisms

No meta mechanism was needed for the simple polynomial problem. For the trigonometric problem, where the value s is important for solving the problem, the system was provided with meta-constants.

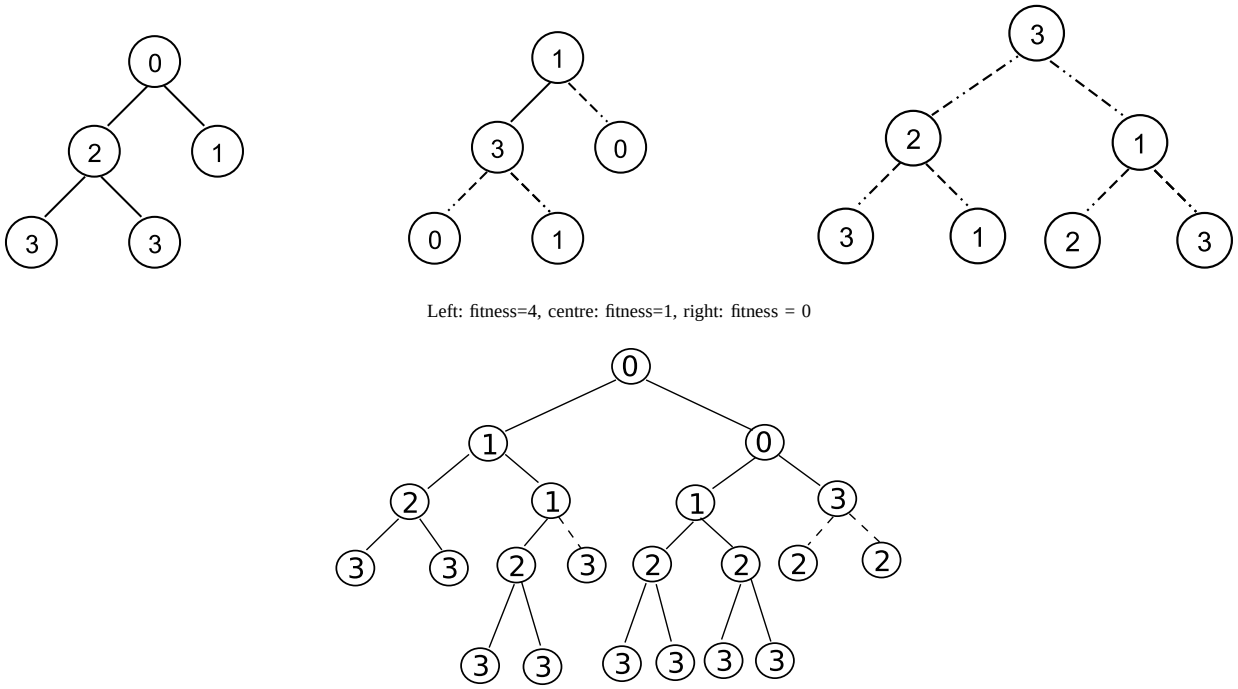
In the polynomial and trigonometric problem families, there is only one domain variable, x. By contrast, in the Boolean n-parity and ORDERTREE problems, each new $(s+1)$ problem introduces a new variable x_{s+1} that cannot be meaningfully evaluated in the preceding s problem. In these problems, we have provided meta-variables to eliminate the difficulty.

C. Comparisons Between Different GP Systems

The first stage of the experiments compared three different systems: GP (a basic Koza-style tree-based GP system [6]); DTAG3P (DTAG - the system described above), and TAG3P (a tree-based GP system, using as its genotype the same TAG representation as DTAG3P's intermediate phenotype, but using an evolutionary process identical to that of GP). The three systems were evaluated on the four problem families from Subsection VI-A.

1) Parameter Settings: To evaluate their performance, we used a fixed budget of function evaluations (num_{eval}), with the population size max_{pop} = 250, and the max generation size being adjusted to maintain this budget. This is important for fair comparison, because DTAG3p uses multiple staged evaluations; thus over max_{gen} generations, it will conduct $num_{eval} = max_{gen} \cdot max_{life} \cdot max_{pop}$ function evaluations. Hence for a fair comparison, the GP and TAG systems should be allowed to run for $max_{gen} \cdot max_{life}$ generations.

The detailed parameter settings are shown in Table IV. The GP and TAG3P settings are typical for these problems. The DTAG3P settings were found by trial and error. We found that high probabilities for the more disruptive operators (rule exchange and rule interchange) led to poor performance, but that performance was insensitive to the rates of other operators. The DTAG3P lifetime was determined by the problem definition (the number of developmental stages must equal the number of problem layers. Other parameters (especially, n_{rules} and max_β), were also determined by experiment. With the exception of δ and n_{pred}, they were not very sensitive, and any reasonable values work. n_{pred} needs to be determined by preliminary experiments for each problem (a value of 1 is a reasonable starting point, but it may not work in all cases, depending on how fast the solution complexity needs to increase with problem layer / developmental stage). The



Left: fitness=4, centre: fitness=1, right: fitness = 0

Fig. 9. 3-ORDERTREE Examples – top: trees of various fitnesses; bottom: an optimal tree. The nodes shown under broken links act as introns.

TABLE IV
EVOLUTIONARY PARAMETER SETTINGS
WHERE DIFFERENT, VALUES FOR DIFFERENT PROBLEMS SEPARATED BY SLASHES.

	GP	TAG3P	DTAG3P
Problems	Polynomial/Trigonometric/Parity/ORDERTREE		
Selection	Tournament, size 3		
Recombination	Rule, Subtree Exchange		
Mutation	Rule Interchange, Subtree and Lexical Mutation		
♯ of Runs			30
num_{eval}			227,250 / 1,359,000 / 202,000 / 126,250
max_{pop}			250 / 1000 / 250 / 250
max_{gen}	909 / 1359 / 808 / 505		101 / 151 / 101 / 101
max_{depth}	30	N/A	N/A
max_{size}	N/A	1,000	NA
p_X		0.9	N/A
p_{mut}		0.1	N/A
p_{copy}		0.1	N/A
max_{life}		N/A	9 / 9 / 8 / 5
n_{rules}		N/A	12
min_{β}		N/A	1
max_{β}		N/A	7
n_{pred}		N/A	1 / 1 / 1 / $\leq$ 4
p_{adapt}		N/A	1.0
p_{good}		N/A	0.05
p_{RX}		N/A	0.1
p_X		N/A	0.24
p_{RI}		N/A	0.1
p_{sub}		N/A	0.24
p_{copy}		N/A	0.08
δ		N/A	0.001 / 0.001 / 0.001 / 0.5

tolerance parameter δ was particularly sensitive, and needed to be separately determined for each problem domain (it undoubtedly depends on the problem; in particular, it should almost certainly depend on the scale of fitness values). We examine the importance of correctly setting δ in our experiments reported below. However we have not examined the parameter settings for the DTAG3P system in detail; we discuss this issue more fully in Subsection IX-C.

2) Problem Grammars: The elementary TAG tree set for the polynomial problem solution space (used inTAG3P and DTAG3P) was previously presented at the top-left of Figure 6.[5]

Since TAG grammars may be unfamiliar to many readers, we also present in Table V the equivalent CFG, G_1.

TABLE V
CFG FOR POLYNOMIAL SYMBOLIC REGRESSION PROBLEM

G_1		=	(V_1, T_1, P_1, S_1)
S_1		=	EXP
V_1		=	{EXP,PRE,OP,VAR}
T_1		=	{x,sin,cos,lg,ep,+,-,*,/}
P_1		=	
	EXP	$\rightarrow$	EXP OP EXP | PRE EXP | VAR
	OP	$\rightarrow$	$+ \mid - \mid * \mid /$
	PRE	$\rightarrow$	sin | cos | lg | ep
	VAR	$\rightarrow$	x

The grammar for the trigonometric problem is identical to that in Figure 6 with two exceptions

1) The unary operators cos, ep and lg are not available as lexical elements (that is, $\beta_{10} \ldots \beta_{12}$ are omitted from the grammar)
2) The variable X is supplemented by a second lexical item, the constant ONE (that is, every remaining β tree, together with the α tree, has a duplicate in which the variable X is replaced by the constant ONE)

This corresponds to the CFG G_2 seen in Table VI.

[5]The operator / is protected division (i.e. returns 1 when the denominator is 0), ep is the exponential function, and lg is the protected logarithm function (returns the log of the absolute value, but returns 0 for the input 0).

TABLE VI
CFG FOR TRIGONOMETRIC SYMBOLIC REGRESSION PROBLEM

G_2		=	(V_2, T_2, P_2, S_2)
S_2		=	EXP
V_2		=	{EXP,PRE,OP,VAR}
T_2		=	{x,ONE,sin,+,-,*,/}
P_2		=	
	EXP	$\rightarrow$	EXP OP EXP | PRE EXP | VAR
	OP	$\rightarrow$	+ | - | $*$ |
	PRE	$\rightarrow$	sin
	VAR	$\rightarrow$	x | ONE

For the k-parity problem, we use the TAG grammar in Figure 10, corresponding to the CFG G_3 in Table VII.

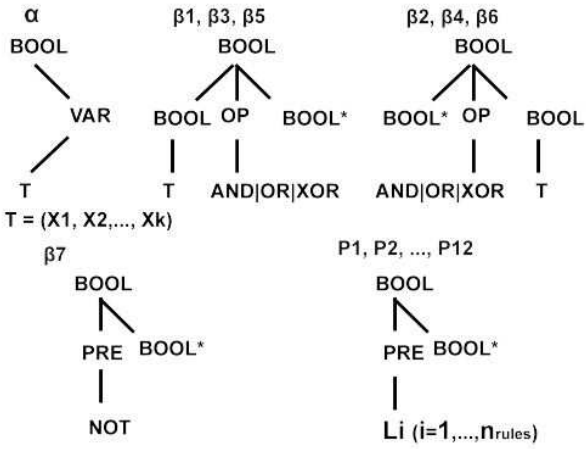


Fig. 10. TAG Elementary Trees for k-Parity Problem

TABLE VII
CFG FOR k-PARITY PROBLEM

G_3		=	(V_3, T_3, P_3, S_3)
S_3		=	EXP
V_3		=	{EXP,PRE,OP,VAR}
T_3		=	$\{x_1, x_2, \ldots, x_k\}$
P_3		=	
	EXP	$\rightarrow$	EXP OP EXP | PRE EXP | VAR
	OP	$\rightarrow$	AND| OR | XOR
	PRE	$\rightarrow$	NOT
	VAR	$\rightarrow$	$x_1, x_2, \ldots, x_k$

For the ORDERTREE problem of order k, the CFG is G_4 shown in Table VIII, resulting from TAG elementary trees corresponding to those in Figure 10, but with the lexicon AND | OR | NOT in the first and second kinds of β trees, and the lexicon $x_1, x_2, \ldots, x_k$ in the fourth, replaced by the lexicon $1, 2, \ldots, k$, and with the third kind of β tree omitted.[6]

D. The Importance of DTAG3P Components

By comparison with standard GP, DTAG3P introduces four major innovations: the representation (TAG trees), the developmental process, layered learning (developmental evaluation)

[6]In this grammar, functions and terminals are both labelled by numbers, the functions always having arity 2.

TABLE VIII
CFG FOR ORDERTREE PROBLEM

G_4		=	(V_4, T_4, P_4, S_4)
S_4		=	EXP
V_4		=	{EXP,OP,VAR}
T_4		=	$\{x_1, x_2, \ldots, x_k\}$
P_4		=	
	EXP	$\rightarrow$	EXP OP EXP | VAR
	OP	$\rightarrow$	$1\|2\|\ldots\|n$
	VAR	$\rightarrow$	$1\|2\|\ldots\|n$

and developmental feedback (meta mechanisms). We argued in the introduction for synergy between these components: that the interaction between these components would bring greater benefits than the individual components. To evaluate the effects of the first three (we plan to study the fourth comprehensively in future work), we introduce three further 'intermediate' treatments. These treatments combine some, but not all, of these components. The systems are:

- *GPgen*: This treatment is designed to address the issue that good performance exhibited by DTAG3P might arise simply from the layered learning process – the increasing difficulty of fitness functions – independent of the developmental process. In this treatment, F_1 is used for the first max_{gen} evaluations, then F_2 and so on; more formally, for i from 0 to n-1, generation (i*max_{gen}) to generation (i+1)*max_{gen} - 1 uses fitness function F_{i+1}. Otherwise, the treatment is identical to GP.
- *TAGgen*: This treatment addresses the hypothesis that the performance of DTAG3P arises from a synergy between layered learning and the TAG representation, and the developmental process is incidental.
- *DTAGFn_all*: This treatment addresses the converse issue, that DTAG3P performance might arise simply from the developmental mechanism having an opportunity to find small solutions, without any need for changing fitness functions (i.e. for layered learning). This treatment uses the multi-stage evaluations of DTAG, but each stage is evaluated using the fitness function F_n, instead of varying through the family F_1 to F_n.

The three treatments above were tested on all four problem domains: polynomial symbolic regression, trigonometric symbolic regression, boolean k-parity, and ORDERTREE.

VII. RESULTS

We examine the performance of the three systems (GP, TAG3P and DTAG3P) on each of the four problem families, comparing them with three other treatments (GPgen, TAGgen and DTAGFn_all) intended to illuminate the behaviour of DTAG3P. Overall success rates are summarised in Table IX.

Figure 11 depicts the cumulative frequencies of success of the various systems (systems not achieving any successes are omitted from the figure for reasons of clarity).

From both tabular and figure data, we can immediately see that only DTAG3P performed well on all these problems. GP and TAG3P were somewhat successful on 6-ORDERTREE (though much less so than DTAG3P), and DTAGFn-all performed acceptably on 8-parity; apart from these, all systems

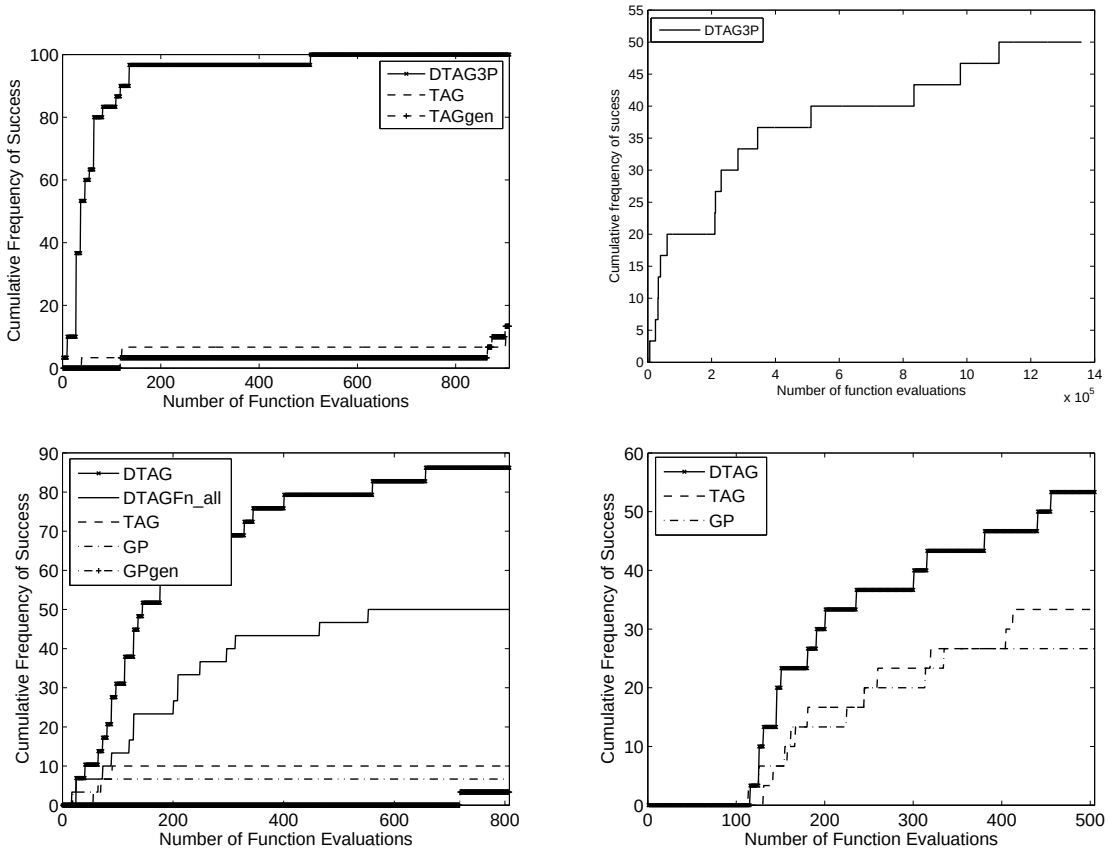


Fig. 11. Cumulative Frequencies of Success of all Systems on Problem Families
Top: Left: Polynomial ($F_i(x)$) and Right: Trigonometric ($\Phi_i(x)$) Symbolic Regression Bottom: Left: k-parity and Right: n-ORDERTREE

TABLE IX
SUCCESS RATES ON POLYNOMIAL (POLY) AND TRIGONOMETRIC (TRIGO) SYMBOLIC REGRESSION, 8-PARITY AND 6-ORDERTREE (6-ORDER)

	POLY	TRIGO	8-PARITY	6-ORDER
DTAG	100%	53.33%	86.67%	53.33
DTAGFn_all	0%	0%	50%	0%
TAG	10%	0%	10%	33.33%
TAGgen	13.33%	0%	0%	0%
GP	0%	0%	6.67%	26.67%
GPgen	0%	0%	3.33%	0%

other than DTAG3P were uniformly unsuccessful on these problems. This is not surprising – they are tough problems.

Our hypothesis about the behaviour of DTAG3P is not merely that it performs well, but that it learns in a layered, incremental fashion. Figure 12 elucidates this further, showing the cumulative success of DTAG3P on the layered subproblems from which (we posit) it builds its overall solutions. In all cases, DTAG3P learnt in a layered, incremental fashion, using the solutions of simpler problems as stepping stones to solutions to the larger-scale problems.

Cumulative frequency of success, however, does not tell the whole story. It is important, also, to see the change in fitness during a run. Figure 13 shows the median (over all runs) of the best fitness for the three main treatments (GP, TAG3P, DTAG3P).[7] Since the fitness scales for the two symbolic regression problems are very compressed, we also show a more expanded view in Figure 14. All confirm that, while GP and TAG3P show better performance early (because DTAG3P at that stage is concentrating on the simpler problems in the family, so its performance on the final problem is essentially random), their performance stagnates, while that of DTAG3P continues to improve, eventually providing much better performance.

Further understanding may be gained from seeing the median performance in each generation. Figure 15 shows these, for each of the three main treatments. Again, in most cases, we see a similar behaviour, with DTAG3P displaying poor early performance, but eventually yielding far better results than the other methods. We carried out similar studies on the other treatments (GPgen, TAGgen, DTAGFn_all), and saw very similar results in all cases.

A. Sensitivity to δ

DTAG3P introduces a number of new parameters; most are not particularly novel (rates of applications of different operators etc.) so that previous experience in evolutionary computation can guide their setting. However one, δ, is

[7]To avoid results being heavily skewed by outliers, all results reported here use the median rather than the mean.

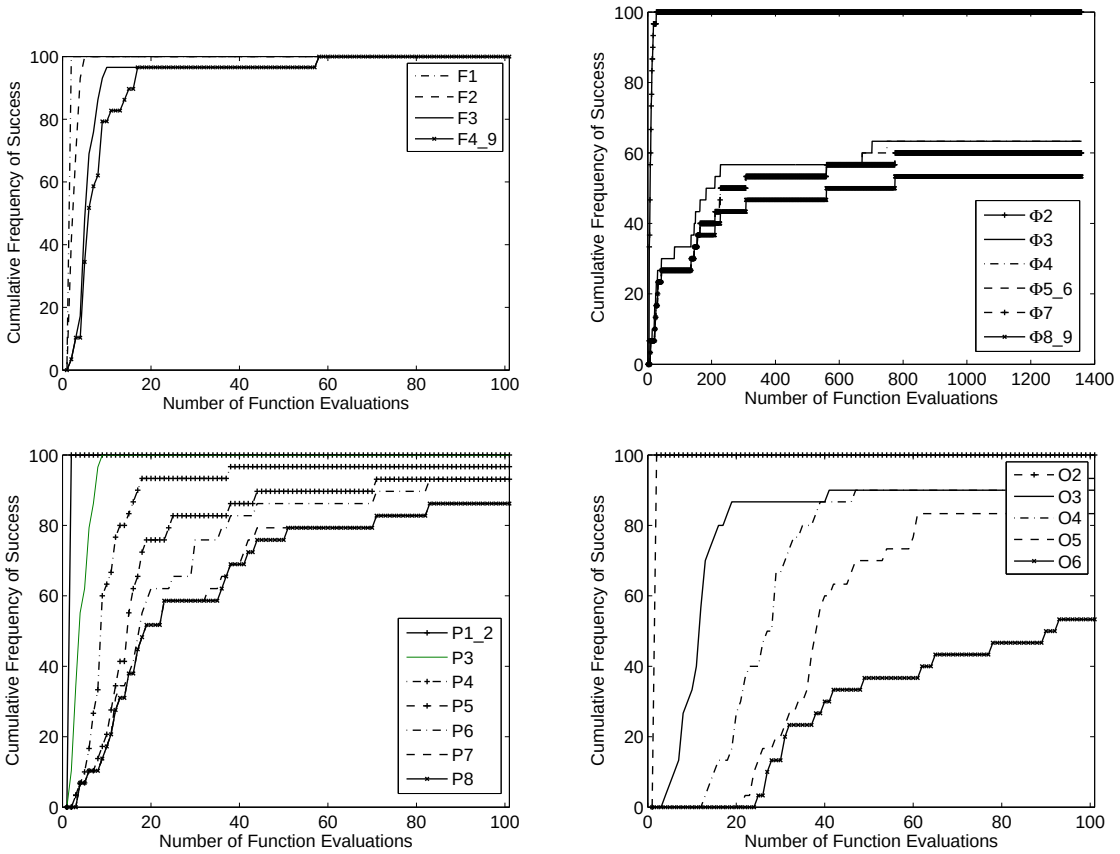


Fig. 12. Cumulative Frequencies of Success of DTAG3P on Problems in Problem Families
Top: Left: Polynomial ($F_9(x)$) and Right: Trigonometric ($\Phi_9(x)$) Symbolic Regression Bottom: Left: 8-parity and Right: 6-ORDERTREE

TABLE X
SUCCESS RATES OF DTAG3P WITH VARYING δ

	δ=0.000001	δ=1.0	δ=5.0
Polynomial	86.67%	100%	40%
Trigonometric	53.33%	63.33%	33.33%

completely new, since it directly relates to the structure of the system. It is also a core parameter in our system, since it controls the interaction between the developmental process and the problem layers.

We performed a test of its effect on the performance of DTAG3P (on the two symbolic regression problems only), varying it over a wide range. Our previous discussion of layered tournaments suggested that too small a value could lead DTAG3P to overfit to simple problem layers at early stages of development, rendering it difficult to recover and build performance on more difficult problem layers. On the other hand, too large a value might also damage performance, by failing to ensure good performance at any level. This is what we found: δ needs to have an intermediate value.

At the moment, we have no a-priori way to predetermine a suitable value. The value of δ for a new problem family should be determined by preliminary experiments. The results in table X suggest that a bias toward smaller values might save time, since the deterioration in performance with small values of δ was much less than with large values.

However it is clear that this area warrants further investigation. Since we have no particular reason to expect the optimal value of δ to be the same at each developmental stage and problem layer, adaptive mechanisms would be worth exploring.

VIII. SIMPLICITY AND REGULARITY

In the preceding analysis, we concentrated on performance issues: how well did DTAG3P solve problems. But in our initial discussion, we hypothesised that developmental evaluation would result not merely in better and more scalable solutions, but better structured, more regular ones. Did this occur?

A. Solution Visualisation

One way to determine this is to examine the solutions by eye. Figure 16 represents a typical solution to the 8-parity problem, as found by DTAG3P. Anyone familiar with GP solutions will be immediately struck by two things: its small size (19 nodes – typical solutions found by GP and TAG respectively contained around 200 and around 1,000 nodes) and lack of introns; and its elegance and repeated regular patterning. In the corresponding D0TL system, we can directly find an encoding of the recurrent expression x_i XOR F_{i-1}.

As in biology, we see the emergence of repeated duplications of structure, though nothing in the system directly

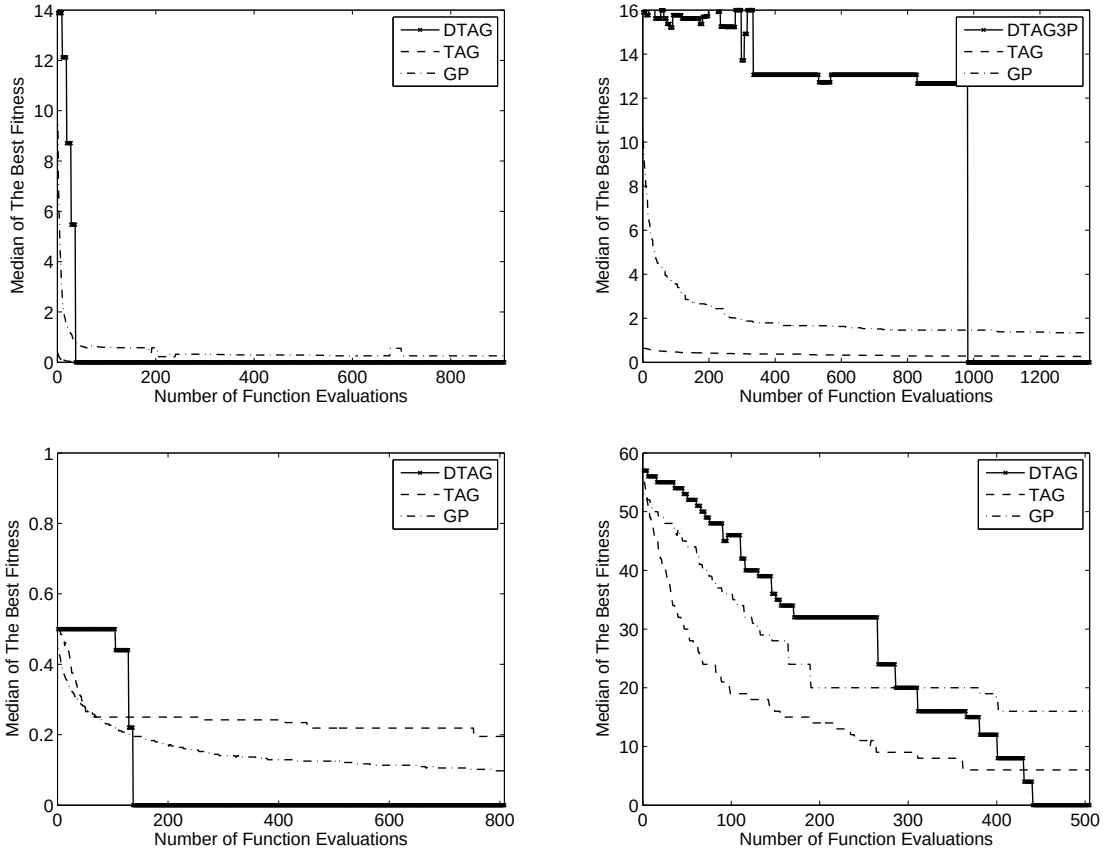


Fig. 13. Median of the Best Fitness
Top: Left: Polynomial ($F_9(x)$) and Right: Trigonometric ($\Phi_9(x)$) Symbolic Regression Bottom: Left: 8-parity and Right: 6-ORDERTREE

requires such duplication. This systematic duplication and regular structure doesn't appear in these systems unless they incorporate the delicate interplay of layered learning and development provided by the layered tournament.

TABLE XI
AN EXAMPLE OF A SOLUTION OF DTAG3P ON THE POLYNOMIAL SERIES: D0TL-REPRESENTATION

P_1: $L_1 \rightarrow \beta_3$:0(β_2:1(L_5:1))
P_2: $L_2 \rightarrow \beta_{10}$:0($\beta_{10}$:0($\beta_{10}$:0($\beta_7$:0($\beta_2$:0($L_{12}$:1)))))
... ...
P_5: $L_5 \rightarrow \beta_6$:0(L_{12}:0,β_1:1)
... ...
P_{10}: $L_{10} \rightarrow \beta_2$:0($L_2$:0)
P_{11}: $L_{11} \rightarrow \beta_2$:0($\beta_4$:1($\beta_3$:0($\beta_2$:0($L_{11}$:1))))
P_{12}: $L_{12} \rightarrow \beta_6$:0($L_{12}$:0,$\beta_1$:1)

Let us look further at this, by examining in detail a solution found by DTAG3P on the polynomial problem family. Table XI shows the key part of the D0TL rules of the genotype, while Table XII depicts the corresponding genotypes and phenotypes in linear string format for development stages 1, 2, 3 and 9. These tables refer to the grammar from Figure 6, which has only one initial tree, (α_0).

Figures 17, 18 and 19 show the corresponding intermediate and final phenotypes: TAG derivation trees, CFG derived trees and GP expression trees. In the TAG tree, we see the regular pattern: β_6:0(L_{12}:0,β_1:1). This results in the circled structure in the CFG tree, which is then repeated into larger and larger trees as development proceeds (the stage 9 tree in this case is too large to draw). Finally, in the expression tree, we see the repeated regular pattern $x + x \cdot F_i$, i.e. the appropriate recurrence relation.

Similar analyses have been conducted for other problems, with similar results, but are omitted here for brevity.

B. Computational Effect of Simplicity and Early Death

In the computational experiments, we used the same number of function evaluations to compare the different GP systems. But was this really fair? Firstly, in equilibrating DTAG3P's computational cost with the other systems, we assumed that DTAG3P developed every individual to its final stage. But in fact, the tournament evaluation that DTAG3P uses does not require this; if lazy evaluation is used, less fit individuals are unlikely to be evaluated through all stages – in effect, they 'die early'. Second, because DTAG3P individuals are much simpler than their competitors, they may cost less to evaluate. In most GP settings, evaluation cost is directly proportional to individual size (or at minimum, to the size of the components actually executed). We decided to take full account of this issue, by comparing the total number of evaluations of expression nodes used in the fitness comparisons. Table XIII shows the result for the parity problem; even though DTAG3P found many more solutions than the other systems, it found

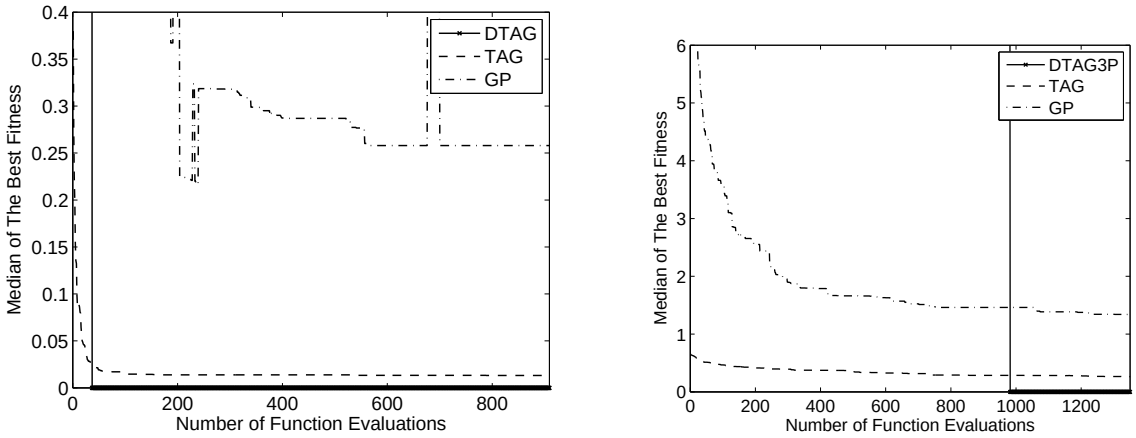


Fig. 14. Close-up of Median of the Best Fitness Left: $F_9(x)$, Right: $\Phi_9(x)$

TABLE XII
INTERMEDIATE AND FINAL PHENOTYPES OF AN EXAMPLE DTAG3P SOLUTION FOR POLYNOMIAL PROBLEMS

S_1	TAG	α_0:0(β_3:0(β_2:1(L_5:1)))
	Trees	
	CFG	EXP(PRE(L_5),EXP(EXP(EXP(VAR(X)),OP(sub),EXP(VAR(X))),OP(add),EXP(VAR(X))))
	Trees	
	GP	add(sub(X,X),X)
	Trees	
S_2	TAG	α_0:0(β_3:0(β_2:1(L_5:1(β_6:0(L_{12}:0,β_1:1)))))
	Trees	
	CFG	EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_5),EXP(EXP(EXP(VAR(X)),OP(sub),EXP(VAR(X))),
	Trees	OP(add),EXP(VAR(X)))),OP(mul),EXP(PRE(L_{12}),EXP(VAR(X)))))
	GP	add(X,mul(add(sub(X,X),X),X))
	Trees	
S_3	TAG	α_0:0(β_3:0(β_2:1(L_5:1(β_6:0(L_{12}:0(β_6:0(L_{12}:0,β_1:1)),β_1:1)))))
	Trees	
	CFG	EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_5),EXP(EXP(EXP(VAR(X)),OP(sub),EXP
	Trees	(VAR(X))),OP(add),EXP(VAR(X)))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP
		(PRE(L_{12}),EXP(VAR(X))),OP(mul),EXP(PRE(L_{12}),EXP(VAR(X)))))))
	GP	add(X,mul(add(sub(X,X),X),add(X,mul(X,X))))
	Trees	
S_9	TAG	α_0:0(β_3:0(β_2:1(L_5:1(β_6:0(L_{12}:0(β_6:0(L_{12}:0(β_6:0(L_{12}:0(β_6:0
	Trees	
		(L_{12}:0(β_6:0(L_{12}:0(β_6:0(L_{12}:0(β_6:0(L_{12}:0,β_1:1)),β_1:1)),β_1:1)),
		β_1:1)),β_1:1)),β_1:1)),β_1:1)),β_1:1)))))
	CFG	EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_5),EXP(EXP(EXP(VAR(X)),OP(sub),EXP
	Trees	(VAR(X))),OP(add),EXP(VAR(X)))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP
		(PRE(L_{12}),EXP(VAR(X))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE
		(L_{12}),EXP(VAR(X))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_{12}),EXP
		(VAR(X))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_{12}),EXP(VAR
		(X))),OP(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_{12}),EXP(VAR(X))),OP
		(mul),EXP(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_{12}),EXP(VAR(X))),OP(mul),EXP
		(EXP(VAR(X)),OP(add),EXP(EXP(PRE(L_{12}),EXP(VAR(X))),OP(mul),EXP(PRE
		(L_{12}),EXP(VAR(X))))))))))))))))))
	GP	add(X,mul(add(sub(X,X),X),add(X,mul(X,add(X,mul(X,add(X,mul(X,add(X,mul
	Trees	(X,add(X,mul(X,add(X,mul(X,add(X,mul(X,X)))))))))))))))

them at far lower computational cost – an order of magnitude less, except in the case of DTAGFn_all.

TABLE XIII
MEAN NUMBER OF NODE EVALUATIONS PER RUN

	8-Parity
DTAG	9294835.3
DTAGFn_all	19066168.8
TAG	142900418.1
TAGgen	84761732.3
GP	208319791.0
GPgen	124944247.5

C. Regularity of Solutions

The solutions found by DTAG3P are smaller, and certainly look simpler, than those found by other systems. But are they really more regular? To answer this question, we made use of the compression metrics briefly outlined in Subsection IV-C, and fully detailed in [88]. We applied these metrics to the output of four of these treatments (GP, TAG3P, DTAG3P and DTAGFn_all). For the purposes of fair comparison, it is important to compare like with like; fortunately, all these systems generate GP expression trees as their final phenotype, so we can directly compare the complexity of these expression trees. A second issue also arises in this case; individuals are of different sizes – perhaps different-sized individuals might

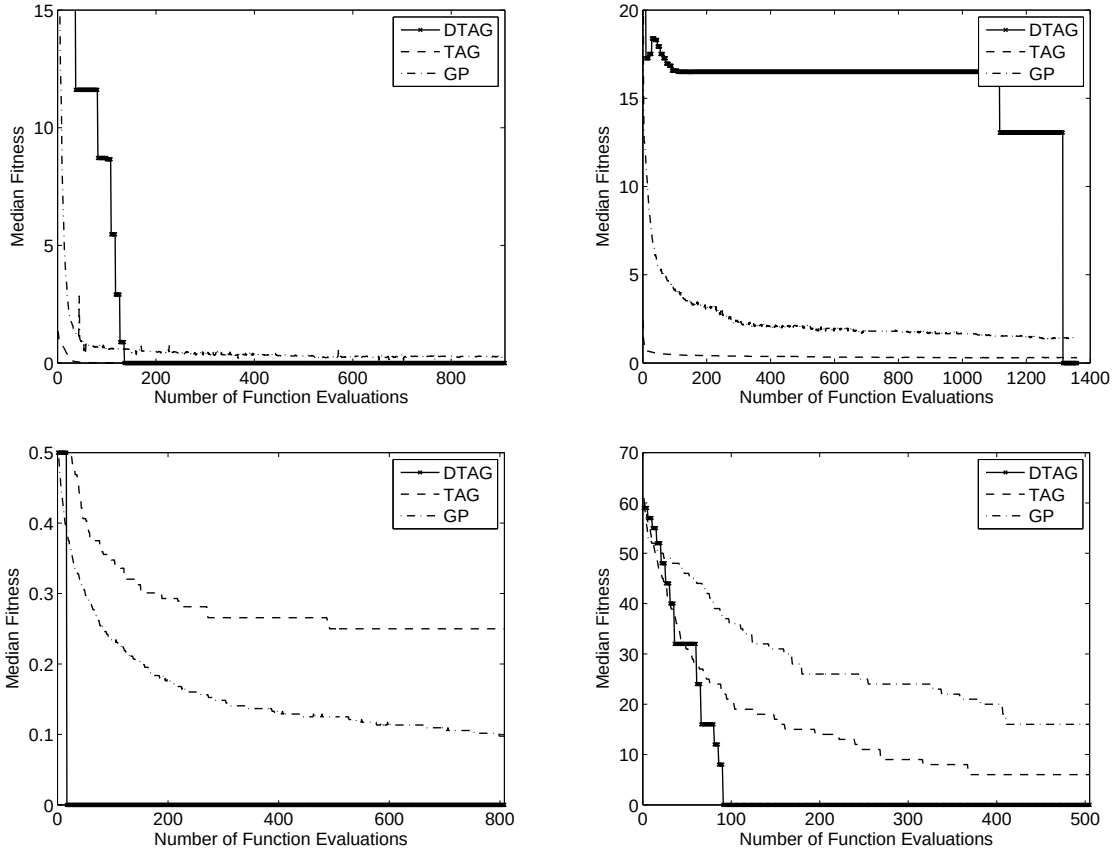


Fig. 15. Median of the Median Fitness
Top: Left: Polynomial ($F_9(x)$) and Right: Trigonometric ($\Phi_9(x)$) Symbolic Regression Bottom: Left: 8-parity and Right: 6-ORDERTREE

be more or less compressible, independent of their regularity. Please see [88] for a detailed discussion of this issue, and of the normalisation mechanisms we have used to overcome it. In the results detailed here, we only look at the results of the polynomial symbolic regression problems, but similar behaviour is found in all the problems studied in this paper.

Figure 20 shows the evolution of individual regularity throughout the runs. We can see, firstly, that DTAG3P generates much higher regularity (larger values indicate higher regularity) than the other systems, whether in the raw code, or to an even greater extent, in the trees simplified to their effective skeleton. DTAGFn_all (shown as DTAGF9_all in the plots) does start off with an initial high degree of regularity (initially bearing out the widely-accepted view that developmental systems, on their own, can promote regularity), but this regularity is rapidly lost. In the end, DTAGFn_all actually generates lower regularity than TAG3P. Standard GP generates the most irregular phenotypes of all.

It should be noted that these metrics rely on the underlying model of the specific compression method – in this case, XMLPPM. Potentially this could be unfair, if XMLPPM was particularly biased to detecting the kinds of regularities that DTAG3P embodies, while ignoring those produced by other systems. This objection is unavoidable, whatever method is used to measure regularity – true regularity metrics, for example based on Solomonoff-Kolmogorov complexity, are uncomputable. But in defence of this issue, we note that XMLPPM was developed for XML compression, with no awareness at the time that it would be applied to the compression of GP trees.

IX. Discussion

A. Synergies between Developmental Components

In our introduction, we proposed that a specific combination of development, evaluation and layered learning might lead to synergies. That is, better and more scalable solutions to families of problems might result, with the solutions being both simpler and more regularly structured. In the event, this has been borne out. As we saw in Section VII, DTAG3P is able to reliably solve problems – symbolic regression for high order polynomials and trigonometric functions, parity problems and ORDERTREE problems – at scales that are far beyond the capabilities of standard GP or of TAG3P. Section VIII showed that it is able to do so while using an order of magnitude less computational resources (measured in node evaluations). It generated solutions that were far smaller, and also more regular and structured. It is also worth noting, in passing, that as a by-product, DTAG3P gave us, effectively for free, solutions to the earlier problems found during layered learning. For example, 86.66% of runs gave us solutions to the 7-parity problem, which is also difficult for TAG and GP.

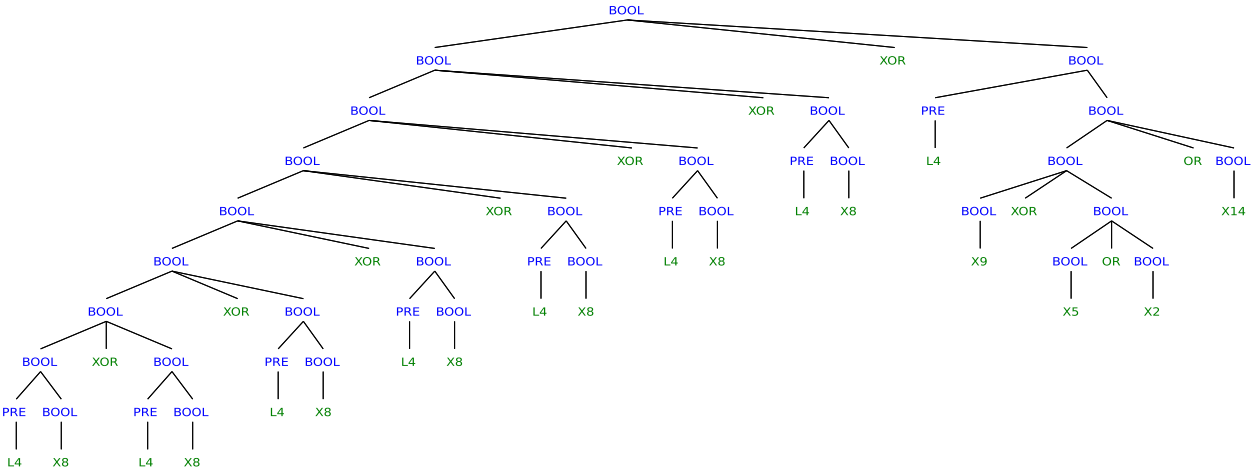


Fig. 16. Expression tree representation of a DTAG3P Solution to 8-parity

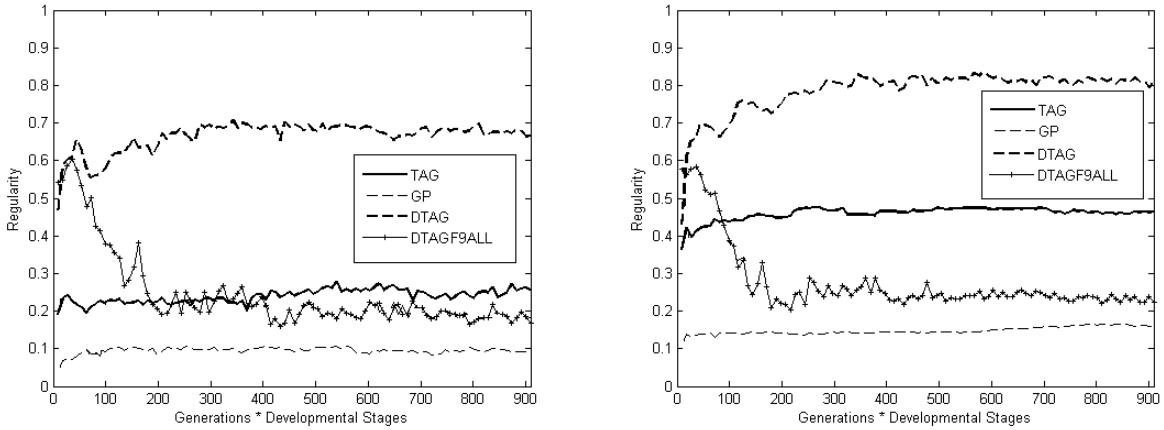


Fig. 20. Individual Complexity vs Function Evaluations: Raw (left) and Simplified (right) Trees

Our other experiments in Section VII showed that these results were not the result of the TAG representation, or of layered learning, or of both in combination, nor were they purely the result of the developmental process or the D0TL genotype representation. The combination of all these components was required to solve these problems, at least in the contexts proposed in this work.

Overall, we believe we have validated our main argument. That is, that an organism which overfits to the problem at one stage will find it difficult to adapt later, and will be out-competed by more adaptable individuals – organisms that, in accordance with parsimony theory, will on average be simpler and more regular.

In this work, we did not test whether the main regularity-promoting mechanism we identified in biology (duplication) would have such an effect in DTAG3P. The main reason was lack of need: regularity emerged (as we predicted) without any explicit regularity-promoting operator. Adding duplication operators would muddy the waters. An additional reason, since it would be an interesting exercise to compare the system with and without duplication operators, is programming complexity. Adding duplication operators would make an already-complex program unwieldy (besides requiring further operator rate parameters). However we note that in other work, we have demonstrated the effectiveness of duplication operators in the non-developmental TAG3P system [103].

B. Assumptions and Limitations

The DTAG3P system proposed here has a number of potential limitations.

Foremost is the layered learning requirement for a family of related problems, rather than a single problem. DTAG3P, and indeed the whole program advanced here, is intimately entwined with layered learning. When layered problems are not available, it is inapplicable. This is really only an issue if DTAG3P is viewed as a competitor for GP. We don't see it this way. For the many problems naturally present without layering; DTAG3P is simply inapplicable. However many other real-world problems have natural layered structures. Examples range from robot behaviour to multi-agent learning, from visual processing to circuit design or robot soccer [37], [38], [40], [41], [104]. For these problems, DTAG3P can take advantage of the layered structure much more effectively than GP or related systems.

Second is its complexity. However the core ideas – of combining layered learning with development – are indepen-

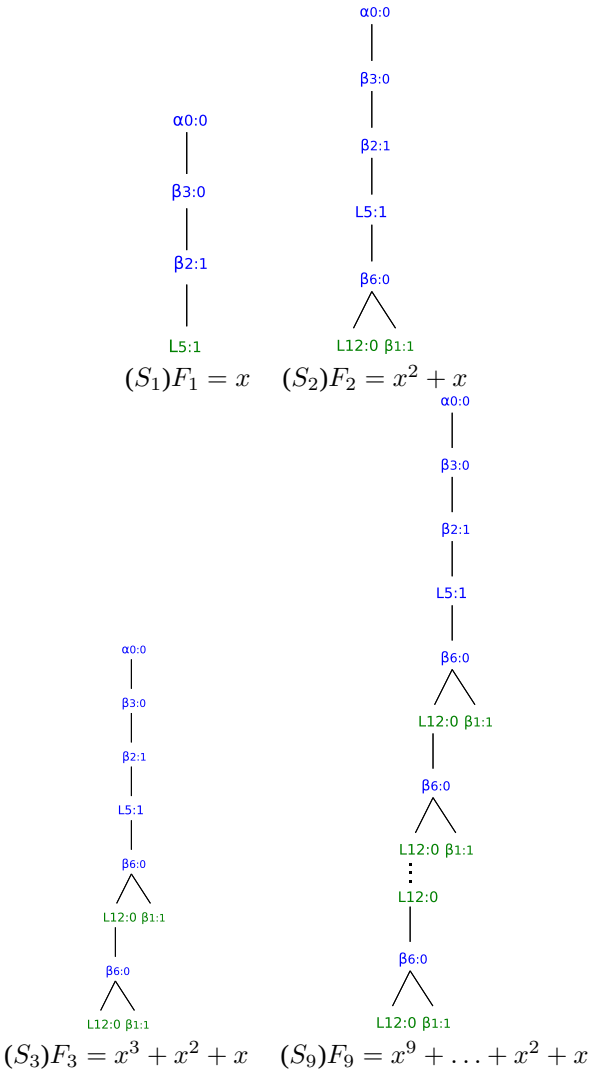


Fig. 17. TAG Tree Solution for Polynomial Series

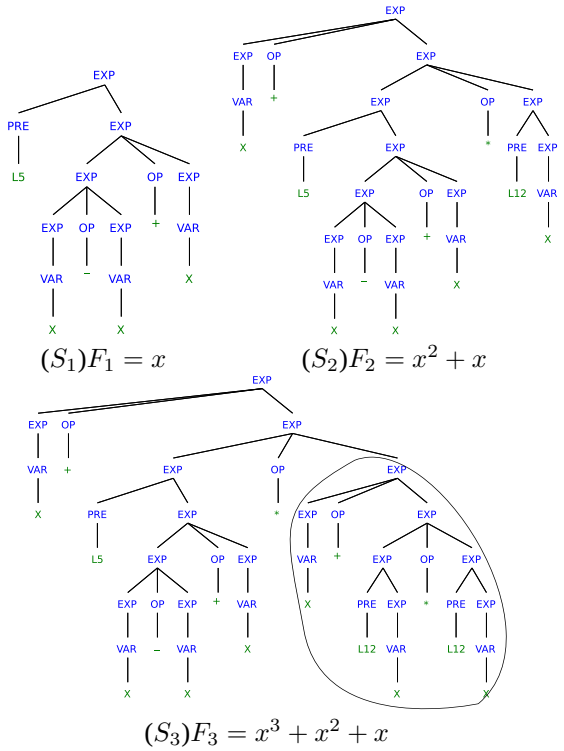


Fig. 18. CFG Tree Solution for Polynomial Series

dent of the particular implementation. For instance, in recent work, McPhee et al. [25] adopted a similar approach for their linear GP system with N-gram probabilistic learning. Their results were consistent with the findings in this paper, in that the combination of layered learning, with evaluation during the developmental process, was vital for the sucess of their DGP system. More generally, these ideas could equally readily be implemented in any other system that could support the requirements of Subsection II-C; Grammatical Evolution [105], Cartesian Genetic Programming [106] and PushGP [107] spring immediately to mind.

The DTAG3P system introduces a number of new parameters, potentially increasing the cost of initial tuning and the risk of overfitting. While we have not yet carried out detailed parameter sensitivity studies, our experience so far suggest that it is not difficult to choose good values for these parameters. Other work not reported here [108] suggests that overfitting is not one of the failings of DTAG3P.

The effects of the meta mechanism have not been investigated in detail in this study. In the case of the polynomial problem, meta-mechanisms were not used, so that at least some of the good performance of DTAG3P was independent of the use of meta-variables and constants. At the other extreme, meta-variables were crucial to the incorporation of layered learning into the developmental process for parity and ORDERTREE problems. Without them, there was no mechanism for layered learning to take place, so no comparisons with and without meta-variables was feasible (we were unable to get penalty approaches to work, because we could not get the initial population to generate sufficient feasible individuals for evolution to proceed). However it is clear that in some cases, meta-mechanisms do add a great deal to the system – for example, we were unable to find good solutions to the trigonometric problem family without them.

C. Future Extensions

A wide range of extensions of this work are possible.

Most immediately, it is clear that the combination of components, for which we use the name evolutionary developmental evaluation (EDE), could be readily applied to a number of other GP systems. We hope to promote and collaborate with such work in the near term.

This paper has compared DTAG3P with tree-based GP systems, because we wished to concentrate on the effects of evaluation during development, which in turn required us to reduce the effects of differences in search space size, fitness landscape complexity etc. These effects can be very substantial [3], [109]. Despite these risks, we hope to carry out a comparison with other developmental systems, especially linear-GP based ones that can handle the same or similar problems, as soon as is feasible.

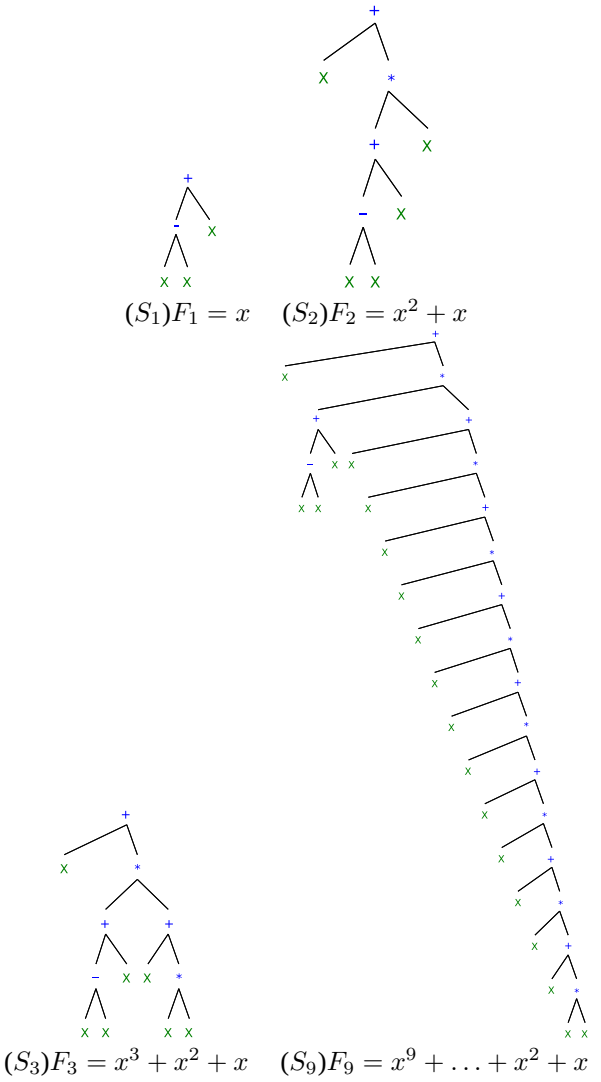


Fig. 19. GP Expression Tree Solutions for Polynomial Series

The meta-mechanism is a prime area for further exploration. Topics of interest include formalisation in terms of higher-order function theory, investigation of a wide range of alternatives for detailed implementation, and broader experimental validation of its value. However the current meta-mechanism lacks a crucial aspect of natural development: the ability of the developmental process to respond to environmental influences. We view this as one of the most promising future research directions.

At a more detailed level, we plan to undertake more detailed studies of parameter sensitivity, especially of the key δ parameter; at the same time, we will be investigating ways to self-adapt these parameters, to remove the tuning load.

In related studies, we are examining the role of DTAG3P as a hyperheuristic, and its ability to learn general solutions, and then adapt them to new problems. Specifically in the area of machine learning, we are currently investigating the potential for developmental evaluation and layered learning to generate more robust, parsimonious solutions to noisy learning problems.

One interesting result of this work is the observation that our D0TL system regularly – in fact, almost universally – evolves recursive grammars. Thus one very interesting possibility is to present DTAG3P with increasing depths of recursion in its problem instances. DTAG3P will hopefully evolve a D0TL system implementing that recursion. With our current mapping strategy, the recursion is not explicitly available at the phenotype level. But once the recursion is explicitly present in the genotype, it should be feasible to use recursion preserving transformations to map the D0TL-level recursion to the genotype level, thus providing a new mechanism for evolving recursive programs.

X. Conclusions

This paper presented a brief survey of current research on modularity and regularity in evolutionary systems, in biology and in artificial life, with emphasis on their role in evolutionary developmental systems and developmental genetic programming systems.

Based on this perspective, we investigated a combination of abstractions from natural mechanisms. We investigated whether this combination, EDE, led to synergies, which produced better performance in combination than as individual components. Specifically, we investigated combinations of:

- Developmental process governed by ‘genes’
- Developmental evaluation
- Evaluation in sequence
- Varying semantics during development (layered learning)
- Adaptive variation rates
- Availability of ”stage” information to the developing organism

In the implementation, we extended the pre-existing Tree Adjoining Grammar Guided Genetic Programming (TAG3P) framework to incorporate these components, resulting in Developmental TAG3P (DTAG3P).

DTAG3P was benchmarked on a range of problems, and compared with conventional non-developmental systems (TAG3P and tree-based GP), and with systems omitting some of the proposed synergistic components.

- EDE was more computationally efficient than its comparators, finding more accurate solutions faster.
- EDE evolved greater regularity, reflecting repeated duplications of segments of the phenome, than did the other systems.
- This regularity resulted in better scalability, which was absent unless all the above components were provided.
- In EDE, parsimony in structure could be preferentially selected without any need for an explicit parsimony management scheme. Subjectively, EDE’s solutions were generally elegant, simple, small in size and easily understandable.
- The evolutionary developmental evaluation mechanism incorporated re-use of building blocks, as confirmed by compression measurements. Thus, it had the ability to explore a larger and more sophisticated problem space by building up from simpler ones.

Overall, the results validated our hypotheses about the effects of combining evolution, lifelong evaluation throughout

development, and layered learning, confirming that the resulting implicit generalisation pressure supported more structured and scalable solutions to problems. It opens up a number of new lines of research, with the potential for significant progress in developing effective, scalable learning systems.

ACKNOWLEDGMENT

We would like to record our thanks to Thi Hien Nguyen, who first drew our attention to the relationship between our methods and layered learning.

This work was supported by a Korea Research Foundation Grant funded by the Korean Government (KRF-2008-313-D00943). The Seoul National University Institute for Computer Technology provided research facilities for this study. The first and fourth authors were partly funded by The Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01.14.09 for doing this work.

REFERENCES

[1] S. Kumar and P. Bentley, Eds., *On Growth, Form and Computers*. Elsevier Academic Press, 2003.

[2] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray, 1858.

[3] 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.

[4] J. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, MI, 1975.

[5] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog Stuttgart, 1973.

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

[7] W. Gilbert, "Origin of life: the RNA world," *Nature(London)*, vol. 319, no. 6055, 1986.

[8] M. Ebner, M. Shackleton, and R. Shipman, "How neutral networks influence evolvability," *Complexity*, vol. 7, no. 2, pp. 19–33, 2001.

[9] P. Whigham, "Grammatically-based genetic programming," in *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, vol. 16, no. 3. Citeseer, 1995, pp. 33–41.

[10] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, 2001.

[11] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems Journal*, vol. 4, pp. 461–476, 1990. [Online]. Available: http://www.complex-systems.com/Archive/volume04/issue4/index.html

[12] F. Gruau, "Genetic synthesis of boolean neural networks with a cell rewriting developmental process," in *Combinations of Genetic Algorithms and Neural Networks, 1992., COGANN-92. International Workshop on*, jun. 1992, pp. 55 –74.

[13] A. Lindenmayer, "Mathematical models for cellular interaction in development," *J. Theoret. Biology*, vol. 18, pp. 280–315, 1968.

[14] 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.

[15] G. S. Hornby and J. B. Pollack, "Evolving L-systems to generate virtual creatures," *Computers and Graphics*, vol. 25, no. 6, pp. 1041–1048, 2001.

[16] 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.

[17] D. Floreano and J. Urzelai, "Neural morphogenesis, synaptic plasticity, and evolution," *Theory in Biosciences*, vol. 120, no. 3, pp. 225–240, 2001.

[18] K. Stanley, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines*, vol. 8, no. 2, pp. 131–162, 2007.

[19] F. Dellaert and R. Beer, "A developmental model for the evolution of complete autonomous agents," in *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, 1996, pp. 393–401.

[20] N. Jakobi, "Harnessing morphogenesis," in *International Conference on Information Processing in Cells and Tissues*. Citeseer, 1995, pp. 29–41.

[21] P. Haddow, G. Tufte, and P. van Remortel, "Evolvable hardware: Pumping life into dead silicon," in *On Growth, Form and Computers*, S. Kumar and P. Bentley, Eds. Elsevier Academic Press, 2003, ch. 22, pp. 405–423.

[22] P. E. Hotz, "Combining developmental processes and their physics in an artificial evolutionary system to evolve shapes," in *On Growth, Form and Computers*, S. Kumar and P. Bentley, Eds. Elsevier Academic Press, 2003, ch. 16, pp. 302–318.

[23] S. Viswanathan and J. Pollack, "How artificial ontogenies can retard evolution," in *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 273–280.

[24] J. F. Miller and W. Banzhaf, "Evolving the program for a cell: from french flags to boolean circuits," in *On Growth, Form and Computers*, S. Kumar and P. Bentley, Eds. Elsevier Academic Press, 2003, ch. 15, pp. 278–301.

[25] N. McPhee, E. Crane, S. Lahr, and R. Poli, "Developmental plasticity in linear genetic programming," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 2009, pp. 1019–1026.

[26] T. S. Ray, "An evolutionary approach to synthetic biology: Zen and the art of creating life," *Artificial Life*, vol. 1, no. 1/2, pp. 195–226, 1994.

[27] S. Young Jung, "A topographical method for the development of neural networks for artificial brain evolution," *Artif. Life*, vol. 11, no. 3, pp. 293–316, 2005.

[28] T. Kowaliw and W. Banzhaf, "Augmenting artificial development with local fitness," in *Congress on Evolutionary Computation*. IEEE, 2009, pp. 316–323.

[29] G. Tufte and P. C. Haddow, "Towards development on a silicon-based cellular computing machine," *Natural Computing: an international journal*, vol. 4, no. 4, pp. 387–416, 2005.

[30] G. Tufte and P. Haddow, "Achieving environmental tolerance through the initiation and exploitation of external information," *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pp. 2485–2492, 2007.

[31] L. Spector and K. Stoffel, "Ontogenetic programming," in *Proceedings of the Conference on Genetic Programming*. MIT Press, 1996, pp. 394–399.

[32] S. Harding, J. Miller, and W. Banzhaf, "Evolution, development and learning using self-modifying cartesian genetic programming," in *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO)*. ACM, 2009, pp. 699–706.

[33] ——, "Self modifying cartesian genetic programming: Parity," in *Proceedings of the Congeress on Evolutionary Computation*, A. Tyrrell, Ed., IEEE. IEEE Press, 2009, pp. 285–292.

[34] T. Mitchell, "Machine learning," *Mac Graw Hill*, p. 368, 1997.

[35] H. De Garis, "Genetic programming: Building nanobrains with genetically programmed neural network modules," in *Neural Networks, IJCNN International Joint Conference on*. IEEE, 1990, pp. 511–516.

[36] P. Stone and M. Veloso, "Layered learning," in *Proceedings of the European Conference on Machine Learning (ECML)*. Springer, 2000, pp. 369–381.

[37] K. Sims, "Evolving 3D morphology and behavior by competition," *Artificial Life*, vol. 1, no. 4, pp. 353–372, 1994.

[38] W. Hsu and S. Gustafson, "Genetic programming and multi-agent layered learning by reinforcements," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2002, pp. 764–771.

[39] D. Jackson and A. Gibbons, "Layered learning in boolean gp problems," in *Proceedings of the European Conference on Genetic Programming (EuroGP)*, ser. Lecture Notes in Computer Science, vol. 4445. Springer, 2007, pp. 148–159.

[40] H. Bolouri, R. Adams, S. George, and A. G. Rust, "Molecular self-organisation in a developmental model for the evolution of large-scale artificial neural networks," in *Proceedings of the International Conference on Neural Informpation Processing and Intelligent Systems (ICONIP)*, S. Usui and T. Omori, Eds., vol. 2, 1998, pp. 797–800.

[41] L. Sekanina and M. Bidlo, "Evolutionary design of arbitrarily large sorting networks using development," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 319–347, 2005.

[42] J. Krohn, P. J. Bentley, and H. Shayani, "The challenge of irrationality: fractal protein recipes for pi," in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2009, pp. 715–722.

[43] R. I. B. 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 (EuroGP 2006)*, ser. Springer Lecture Note in Computer Science, P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekart, Eds. Heidelberg, Germany: Springer-Verlag, April 10-12 2006, vol. 3905, pp. 280–289, book Chapter. [Online]. Available: http://sc.snu.ac.kr/PAPERS/DevTAG.acr.pdf

[44] T.-H. Hoang, D. Essam, R. B. McKay, and N. X. Hoai, "Developmental evaluation in genetic programming: The tag-based frame work," *International Journal of Knowledge-Based and Intelligent Engineering Systems*, vol. 12, no. 1, pp. 69–82, Jan 2008 2008. [Online]. Available: http://sc.snu.ac.kr/PAPERS/kes.pdf

[45] N. L. Cramer, "A representation for the adaptive generation of simple sequential programs," in *Proceedings of an International Conference on Genetic Algorithms and the Applications*, J. J. Grefenstette, Ed., Carnegie-Mellon University, Pittsburgh, PA, USA, 24-26 Jul. 1985, pp. 183–187.

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

[47] A. Hüttenhofer, P. Schattner, and N. Polacek, "Non-coding rnas: hope or hype?" *Trends in Genetics*, vol. 21, no. 5, pp. 289 – 297, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/B6TCY-4FSCMGD-2/2/f432f0aa7ec134ad568a1d2605755de7

[48] J. Ponjavic, C. P. Ponting, and G. Lunter, "Functionality or transcriptional noise? evidence for selection within long noncoding rnas." *Genome research*, vol. 17, no. 5, pp. 556–565, May 2007. [Online]. Available: http://dx.doi.org/10.1101/gr.6036807

[49] M. J. Oliver, D. Petrov, D. Ackerly, P. Falkowski, and O. M. Schofield, "The mode and tempo of genome size evolution in eukaryotes," *Genome Research*, vol. 17, no. 5, pp. 594–601, May 2007. [Online]. Available: http://dx.doi.org/10.1101/gr.6096207

[50] X. Chen, M. Li, B. Ma, and J. Tromp, "DNACompress: fast and effective DNA sequence compression," *Bioinformatics*, vol. 18, no. 12, pp. 1696–1698, 2002.

[51] W. J. Gehring and Y. Hiromi, "Homeotic genes and the homeobox," *Annual Review of Genetics*, vol. 20, no. 1, pp. 147–173, 1986. [Online]. Available: http://dx.doi.org/10.1146/annurev.ge.20.120186.001051

[52] J. F. Ryan, M. E. Mazza, K. Pang, D. Q. Matus, A. D. Baxevanis, M. Q. Martindale, and J. R. Finnerty, "Pre-bilaterian origins of the hox cluster and the hox code: Evidence from the sea anemone, nematostella vectensis," *PLoS ONE*, vol. 2, no. 1, pp. e153+, January 2007.

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

[54] E. D. D. Jong, "Representation development from pareto-coevolution," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-03*, e. In E. Cantu-Paz et al., Ed. Springer, 2003, pp. 262–273.

[55] J. R. Woodward, "Modularity in genetic programming," in *Genetic Programming, Proceedings of EuroGP'2003*, ser. LNCS, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds., vol. 2610. Essex: Springer-Verlag, 14-16 Apr. 2003, pp. 254–263.

[56] H. Lipson, "Principles of modularity, regularity, and hierarchy for scalable systems." in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-04*, e. In E. Cantu-Paz et al., Ed., vol. 3102. Springer Berlin, 2004, pp. 61–66.

[57] ——, "Principles of modularity, regularity, and hierarchy for scalable systems," *The Journal of Biological Physics and Chemistry*, vol. 7, no. 4, pp. 125–128, 2007. [Online]. Available: http://www.amsi.ge/jbpc/40707/40701.html

[58] P. K. Lehre, "Evolved digital circuits and genome complexity," in *EH '05: Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 79–86.

[59] M. K. Richardson and G. Keuck, "Haeckel's abc of evolution and development," *Biological Reviews*, vol. 77, no. 04, pp. 495–528, 2002.

[60] S. J. Gould, Ed., *Ontogeny and Phylogeny*. Harvard University Press, 1977.

[61] B. Hall, "Evo-devo: Evolutionary developmental mechanisms," *International Journal of Developmental Biology*, vol. 47(7-8 Special Issue), pp. 491–495, 2003.

[62] E. Lewis, "A gene complex controlling segmentation in drosophila," *Nature*, vol. 276, pp. 565–570, 1978.

[63] S. B. Carroll, "Homeotic genes and the evolution of arthropods and chordates," *Journal of Nature*, vol. 376, pp. 479–485, 1995.

[64] ——, *Endless Forms Most Beautiful: The New Science of Evo Devo and the Making of the Animal Kingdom*. Norton, 2005.

[65] R. Calabretta, S. Nolfi, D. Parisi, and G. Wagner, "A case study of the evolution of modularity: Towards a bridge between evolutionary biology, artificial life, neuro- and cognitive science," *Understanding the development and evolution of complex natural systems. Modularity*, vol. MA 1998a, pp. 275–284, 1998.

[66] J. A. Bolker, "Modularity in development and why it matters in evo-devo," *Am. Zool.*, vol. 40, pp. 770–776, 2000.

[67] R. Calabretta and D. Parisi, "Evolutionary connectionism and mind/brain modularity." *Understanding the development and evolution of complex natural systems. Modularity*, vol. MA: 2005, pp. 309–330, 2005.

[68] R. Belshaw, O. Pybus, and A. Rambaut, "The evolution of genome compression and genomic novelty in RNA viruses," *Genome Research*, vol. 17, no. 10, p. 1496, 2007.

[69] P. Rajarajeswari1 and A. Apparao, "DNABIT compress – genome compression algorithm," *Bioinformation*, vol. 5, no. 8, pp. 350–360, January 2011.

[70] M. Kellis, B. Birren, and E. Lander, "Proof and evolutionary analysis of ancient genome duplication in the yeast saccharomyces cerevisiae," *Journal of Nature*, vol. 428, pp. 617–624, 2004.

[71] A. Paterson, J. Bowers, M. Burow, X. Drayeb, C. Elsik, C. Jiangb, C. Katsar, T. Lan, Y. Lin, R. Ming, and R. Wright, "Comparative genomics of plant chromosomes," pp. 1523–1540, Sept 2000.

[72] P. Dehal and J. L. Boore, "Two rounds of whole genome duplication in the ancestral vertebrate," *PLoS Biol*, vol. 3, no. 10, p. e314, 09 2005.

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

[74] X. Gu, "Evolution of duplicate genes versus genetic robustness against null mutations," 2003.

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

[76] H. Ellegren and J. Parsch, "The evolution of sex-biased genes and sex-biased gene expression," *Nature Reviews Genetics*, vol. 8, pp. 689–698, 2007.

[77] J. Kreahling and B. R. Graveley, "The origins and implications of aluternative splicing." *Trends in genetics : TIG*, vol. 20, no. 1, pp. 1–4, January 2004.

[78] S. R. Vallente and E. Eichler Evan, "Segmental duplications and the evolution of the primate genome," Royaume-Uni, London, pp. 65–672, 2002.

[79] L. Ben-Tabou and E. Davidson, "Gene regulation: gene control network in development." *Annual review of biophysics and biomolecular structure*, vol. 36, p. 191, 2007.

[80] N. Chomsky, "Three models for the description of language," *IRE Transactions on Information Theory*, vol. 2, pp. 113–124, 1956.

[81] A. Joshi, L. Levy, and M. Takahashi, "Tree adjunct grammars," *Journal of Computer and Systems Sciences*, vol. 21, no. 2, pp. 136–163, 1975.

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

[83] 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.

[84] 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.

[85] 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.

[86] G. S. Hornby, "Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design," in *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, H.-G. Beyer, U.-M. O'Reilly, D. V. Arnold, W. Banzhaf, C. Blum, E. W. Bonabeau, E. Cantu-Paz, D. Dasgupta, K. Deb, J. A. Foster, E. D. de Jong, H. Lipson, X. Llora, S. Mancoridis, M. Pelikan, G. R. Raidl, T. Soule, A. M. Tyrrell, J.-P. Watson, and E. Zitzler, Eds., vol. 2. Washington DC, USA: ACM Press, 25-29 Jun. 2005, pp. 1729–1736.

[87] ——, "Measuring complexity by measuring structure and organization," in *2007 IEEE Congress on Evolutionary Computation*, D. Srinivasan and L. Wang, Eds., IEEE Computational Intelligence Society. Singapore: IEEE Press, 25-28 Sep. 2007, pp. 2017–2024.

[88] R. I. B. McKay, J. Shin, T. H. Hoang, X. H. Nguyen, and N. Mori, "Using compression to understand the distribution of building blocks in genetic programming populations," in *2007 IEEE Congress on Evolutionary Computation*, D. Srinivasan and L. Wang, Eds., IEEE Computational Intelligence Society. Singapore: IEEE Press, 25-28 Sep. 2007, pp. 2501–2508. [Online]. Available: http://sc.snu.ac.kr/PAPERS/cec07.pdf

[89] J. Cheney, "Compressing xml with multiplexed hierarchical ppm models," in *In IEEE Data Compression Conference*, Snowbird, Utah, 2002, pp. 163–172.

[90] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications*, vol. 32, pp. 396–402, 1984.

[91] P. Nordin and W. Banzhaf, "Complexity compression and evolution," in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, L. Eshelman, Ed. Pittsburgh, PA, USA: Morgan Kaufmann, 15-19 Jul. 1995, pp. 310–317. [Online]. Available: ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/papers/icga95-1.ps.gz

[92] W. B. Langdon, T. Soule, R. Poli, and J. A. Foster, "The evolution of size and shape," in *Advances in Genetic Programming 3*, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, Eds. Cambridge, MA, USA: MIT Press, Jun. 1999, ch. 8, pp. 163–190. [Online]. Available: http://www.cs.bham.ac.uk/ wbl/aigp3/ch08.pdf

[93] R. Poli, "General schema theory for genetic programming with subtree-swapping crossover," University of Birmingham, School of Computer Science, Tech. Rep. CSRP-00-16, Nov. 2000. [Online]. Available: ftp://ftp.cs.bham.ac.uk/pub/tech-reports/2000/CSRP-00-16.ps.gz

[94] D. Hooper and N. S. Flann, "Improving the accuracy and robustness of genetic programming through expression simplification," 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, p. 428.

[95] A. Ekart, "Shorter fitness preserving genetic programs," in *Artificial Evolution. 4th European Conference, AE'99, Selected Papers*, ser. LNCS, C. Fonlupt, J.-K. Hao, E. Lutton, E. Ronald, and M. Schoenauer, Eds., vol. 1829, Dunkerque, France, 3-5 Nov. 2000, pp. 73–83. [Online]. Available: http://www.sztaki.hu/ ekart/ea.ps

[96] P. Wong and M. Zhang, "Algebraic simplification of genetic programs during evolution," Computer Science, Victoria University of Wellington, New Zealand, Tech. Rep. CS-TR-06-7, Feb. 2006. [Online]. Available: http://www.mcs.vuw.ac.nz/comp/Publications/archive/CS-TR-06/CS-TR-06-7.pdf

[97] T.-H. Hoang, D. Essam, R. McKay, and H. Nguyen, "Building on success in genetic programming: Adaptive variation and developmental evaluation," in *The Second International Symposium on Intelligence Computation and Applications*, ser. Lecture Notes in Computer Science, L. Kang, Y. Liu, and S. Zeng, Eds., vol. 4683. Springer-Verlag, 2007, pp. 137–146.

[98] W. B. Langdon and R. Poli, "Why "building blocks" don't work on parity problems," University of Birmingham, School of Computer Science, Tech. Rep. CSRP-98-17, 13 Jul. 1998. [Online]. Available: ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1998/CSRP-98-17.ps.gz

[99] R. Poli and J. Page, "Solving high-order boolean parity problems with smooth uniform crossover, sub-machine code GP and demes," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1/2, pp. 37–56, Apr. 2000. [Online]. Available: http://citeseer.ist.psu.edu/335584.html

[100] T.-H. Hoang, N. X. Hoai, N. T. Hien, R. I. McKay, and D. Essam, "ORDERTREE: a new test problem for genetic programming," in *GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, M. Keijzer, M. Cattolico, D. Arnold, V. Babovic, C. Blum, P. Bosman, M. V. Butz, C. Coello Coello, D. Dasgupta, S. G. Ficici, J. Foster, A. Hernandez-Aguirre, G. Hornby, H. Lipson, P. McMinn, J. Moore, G. Raidl, F. Rothlauf, C. Ryan, and D. Thierens, Eds., vol. 1. Seattle, Washington, USA: ACM Press, 8-12 Jul. 2006, pp. 807–814. [Online]. Available: http://www.cs.bham.ac.uk/ wbl/biblio/gecco2006/docs/p807.pdf

[101] J. M. Daida, J. A. Polito, S. A. Stanhope, R. R. Bertram, J. C. Khoo, and S. A. Chaudhary, "What makes a problem GP-hard? analysis of a tunably difficult problem in genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2. Orlando, Florida, USA: Morgan Kaufmann, 13-17 Jul. 1999, pp. 982–989. [Online]. Available: ftp://ftp.eecs.umich.edu/people/daida/papers/GECCO99landscape.pdf

[102] J. Schaffer and L. Eshelman, "On crossover as an evolutionary viable strategy," in *In Proceedings of the 4th International Conference on Genetic Algorithms*, Morgan Kaufmann, 1991, pp. 61–68.

[103] N. X. Hoai, R. I. B. McKay, D. Essam, and H. T. Hao, "Genetic transposition in tree-adjoining grammar guided genetic programming: The duplication operator," in *Proceedings, European Conference on Genetic Programming*, ser. Lecture Notes in Computer Science, M. Keijzer, A. Tettamanzi, P. Collet, J. v. Hemert, and M. Tomassini, Eds., vol. 3447. Springer Berlin / Heidelberg, 2005, pp. 141–141.

[104] V. Ciesielski, D. Mawhinney, and P. Wilson, "Genetic programming for robot soccer," in *RoboCup 2001: Robot Soccer World Cup V*, ser. Lecture Notes in Computer Science, vol. 2377. Springer, 2002, pp. 37–55.

[105] M. O'Neill and C. Ryan, "Grammatical evolution," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 349–358, Aug. 2001.

[106] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Genetic Programming, Proceedings of EuroGP'2000*, ser. LNCS, R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, Eds., vol. 1802. Edinburgh: Springer-Verlag, 15-16 Apr. 2000, pp. 121–132.

[107] L. Spector, J. Klein, and M. Keijzer, "The push3 execution stack and the evolution of control," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*. ACM, 2005, p. 1696.

[108] T. Hoang, R. McKay, D. Essam, and X. Nguyen, "Learning General Solutions through Multiple Evaluations during Development," in *Proeedings, Evolvable Systems: From Biology to Hardware*, ser. Lecture Notes in Computer Science, vol. 5216. Springer, 2008, pp. 201–212.

[109] N. X. Hoai, R. I. B. McKay, D. Essam, and H. Abbass, "Toward an alternative comparison between different genetic programming systems," in *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, ser. LNCS, M. Keijzer, U.-M. O'Reilly, S. M. Lucas, E. Costa, and T. Soule, Eds., vol. 3003. Coimbra, Portugal: Springer-Verlag, 5-7 Apr. 2004, pp. 67–77.

Hoang, Tuan Hao received his BE in Information Technology from Vietnamese Military Technical Academy, Hanoi (also called Le Quy Don Technical University) in 2001 and his MSc from the University of New South Wales at the Australian Defence Force Academy, Australia in 2004, where he subsequently completed his PhD in Computer Science in 2009. He has joined the Faculty of Information Technology, Le Quy Don Technical University, Hanoi, Vietnam since 2001 as a lecturer/researcher. Since 2009, he has also been a visiting researcher at Hanoi University, Vietnam. Hao's research interests include Evolutionary Computation, Genetic Programming, Grammar Guided Genetic Programming, Developmental Systems and Incremental Learning. He has published numerous research papers in these subjects and has served as a program committee member for a number of national, regional academic conferences on Evolutionary Computation and related fields.

Bob McKay received his BSc in Pure Mathematics from the Australian National University in 1971, and his PhD in the theory of computation from the University of Bristol, UK, in 1976.

He was a Research Scientist in computer typesetting at the (Australian) Commonwealth Scientific and Industrial Research Organisation from 1976 to 1985, before joining the University of New South Wales at the Australian Defence Force Academy as a Lecturer and subsequently Senior Lecturer, with research interests in artificial intelligence, evolutionary computation and ecological modelling. In 2005, he was appointed as Associate Professor and subsequently full Professor at Seoul National University, Korea, where he is continuing these interests. In recent years, he has authored over 150 research papers in these subjects.

Bob was General Chair of the Computational Intelligence Society's 2003 Congress on Evolutionary Computation, and currently serves as a member of its Evolutionary Computation Technical Committee. He has chaired a number of national and regional conferences in Evolutionary Computation and Artificial Intelligence. He is an Associate Editor of the Transactions on Evolutionary Computation, an editorial board member of Genetic Programming and Evolvable Machines, and an advisory board member of the International Journal of Knowledge-Based and Intelligent Engineering Systems

Daryl Essam received his BSc from the University of New England in 1990 and his PhD from the University of New South Wales in 2000. He was an associate lecturer at the University of New England from 1991 and has been with the Australian Defence Force Academy campus of the University of New South Wales since 1994, where he is now a senior lecturer.

His research interests include genetic algorithms, with a focus on both genetic programming and multi-objective optimisation.

Nguyen, Xuan Hoai received his BSc in Computer Science from Hanoi University in 1995, his MSc in Mathematics for Computer and Computing Systems from Vietnamese National University in 1997, and his PhD in Computer Science from the University of New South Wales at the Australian Defence Force Academy, Australia in 2005. He was a lecturer at the department of information technology, Vietnamese Military Technical Academy (VMTA) from 1997 to 2000. Upon completing his PhD degree in Australia, he came back to VMTA in 2005, taking a position as a senior lecturer in the Department of Information Technology. From 2007 to 2009, he was a BK Assistant Professor at Seoul National University, the Republic of Korea. Since 2010, he has been a senior lecturer and director of HANU Information Technology Research and Development Center at Hanoi University, Vietnam. Hoai's research interests include Evolutionary Computation and related fields, in particular Genetic Programming, Grammar Guided Genetic Programming, Artificial Immune Systems, and Fractal Theory. He has published numerous academic papers in the fields of evolutionary computation and genetic programming and has served as a program committee member for a number of top academic conferences on evolutionary computation.