Tree-Adjunct Grammatical Evolution

Eoin Murphy, Michael O'Neill, Edgar Galván-López and Anthony Brabazon
Natural Computing Research & Applications Group
University College Dublin
Ireland
{eoin.murphy, m.oneill, edgar.galvan, anthony.brabazon}@ucd.ie

***Abstract*— In this paper we investigate the application of Tree-Adjunct Grammars (TAG) to Grammatical Evolution (GE). The standard type of grammar used by GE, context-free grammars, produce a subset of the languages that TAGs can produce, making TAGs, expressively, more powerful. In this study we shed some light on the effects of TAGs in GE (called Tree-Adjunct Grammatical Evolution (TAGE) in this work). We perform an analytic comparison of the performance of both setups (i.e., GE and TAGE) across a number of classic genetic programming benchmarking problems taken from the specialised literature. The results firmly indicate that TAGE has a better overall performance (measured in terms of finding the global optima).**

I. Introduction

Grammatical Evolution (GE) [2], [24], [3], since its inception, has had much success. A large proportion of this success can be attributed to how easily GE can be extended. Many different grammars have been explored with GE, including shape grammars[25], attribute grammars[1] and logic grammars[15]. In this paper we explore the utility of TAGs in GE.

The goal of this study is to introduce TAGE, which extends standard GE by incorporating TAGs in its operation. We show how the incorporation of TAG in GE translates into a more effective GE to find solutions on a number of problems with very different landscape features.

As outlined in [14], the set of languages produced by CFGs, known as context-free languages (CFLs), are strictly included in the set of languages produced by TAGs, known as tree-adjunct languages (TALs), which in turn are strictly included in indexed languages, which are finally strictly included in context-sensitive languages. Consequently, this indicates that TAGS are the next step for GE in terms of choice of grammar type.

This paper is structured as follows. A brief overview of GE and its genotype-phenotype mapping process is given in the following section, including an introduction into TAGs. The approach taken in this study to utilise TAGs in GE is described in Section III. The paper continues by describing the experimental setup in Section IV, the results and discussion in Section V, before closing with some conclusions and future work in Section VI.

II. Background

A. Grammatical Evolution

GE is a grammar-based approach to Genetic Programming (GP) [16], [26]. GE combines principles from genetics and molecular biology, the genotype-phenotype mapping, with the representational power of formal grammars, the ability to change the behaviour of the algorithm by simply changing the structure of the grammar. The grammar, a CFG which is usually written in Backus-Naur form, can be easily modified to output programs of an arbitrary language, something that is not a trivial task in other forms of GP. In addition to this, GE's genotype-phenotype mapping means that search operators can be applied to the genotype (usually an integer or binary chromosome), as well as the ability to apply standard GP search operations to the phenotype, therefore extending the search capabilities of standard GP.

1) Genotype to Phenotype Map: GE's genotype-phenotype mapping constructs a derivation tree using a chromosome and a grammar; it operates as follows (see Figure 1 for the grammar and chromosome used for this example). Mapping begins with the start symbol, usually the first symbol declared in the grammar, `<e>`. The first codon (or integer value) is read from the chromosome, in this case it is the value `12`. The number of production rules for the start symbol are counted, `2`, `<e><o><e>` and `<v>`. Which rule to choose is decided according to the mapping function `i mod c`, where `i` is the value of the codon read from the chromosome and `c` is the number of choices available, e.g, `12 mod 2 = 0`, therefore we chose the zero-th rule and `<e>` is expanded to `<e><o><e>`. This expansion forms a partial derivation tree with the start symbol as the root, attaching each of the new symbols as child nodes of the root. The next symbol to expand is the first non-terminal leaf node discovered while traversing the tree in a depth first manner. It should be noted that there is an on going study into variations on the method used to choose which node to expand next [21], [22]. This will be the left-most `<e>` in the tree. The next codon is read and has a value of, `3`, expanding this `<e>` to `<v>`, growing the tree further. The next symbol is the `<v>` previously expanded and the next codon has a value of `7`, `7 mod 2 = 1`, so the rule at index 1, `Y`, is chosen, and so on.

This continues until either there are no more non-terminal leaf nodes left to expand, or until there are no codons

```
Chromosome: 12,3,7,15,9,10,14

Grammar:
<e> ::= <e> <o> <e> | <v>
<o> ::= + | -
<v> ::= X | Y
```

Fig. 1. Example context-free grammar and integer chromosome.

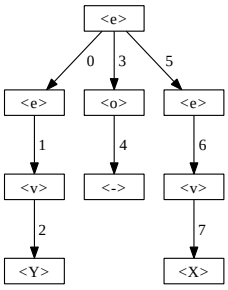


Fig. 2. GE derivation tree.

left to read, i.e., the end of the chromosome has been reached. If there are no codons left to read and derivation has not finished, there are still non-terminal leaf nodes in the derivation tree, derivation can proceed in one of a few different manners. For example, assign a bad fitness to the individual, so it is highly unlikely that it will survive. Another approach can be the use of wrapping. The chromosome maybe reused for a predefined number of times or wrappings. If after the wrapping limit is reached and we still have an invalid individual, we could then assign it a bad fitness.

It should be noted that other approaches might be employed, such as choosing only those rules which produce terminals from the grammar when a specific derivation tree depth [5]. Ensuring that the individuals are valid, i.e., they have no non-terminal leaf nodes. The complete derivation tree for this example is shown in Figure 2 (the labels on the edges indicate the order of expansion).

B. Tree Adjunct Grammars

TAGs, which were introduced first in [13], are a tree generating system. Originally making use of only one composition operation, adjunction. TAGs have since been renamed as tree adjoining grammars and extended to use a second composition operation, substitution. TAGs have been utilised with much success (see [12], [17] and more recently [14]). In particular, in the field of GP in the form of TAG3P [7], [8], [10], [20], [6], [9]. It should be noted that TAGs (only adjunction) and tree adjoining grammars (substitution also) are, formally, as powerful as each other, that is to say, they produced the same set of languages [7]. The inclusion of substitution allows for a more compact formalism, i.e., less trees [14].

This paper is concerned primarily with the original definition of TAGs as outlined in [13], with adjunction as the only composition operation. With this in mind, married with the clearer definition of tree adjoining grammars presented in [14], we present the following definition of TAGs.
A TAG is defined by a quintuple (T, N, I, A, S) where:

1) T is a finite set of terminal symbols;
2) N is a finite set of non-terminal symbols: $T \cap N = \emptyset$;
3) S is the start symbol: $S \in N$;
4) I is a finite set of finite trees. The trees in I are called *initial trees* (or α trees). An initial tree has the following properties:
   - the root node of the tree is labelled with S;
   - the interior nodes are labeled with non-terminal symbols;
   - the leaf nodes, or the nodes along the frontier are labeled with terminal symbols;
5) A is a finite set of finite trees. The trees in A are called *auxiliary trees* (or β trees). An auxiliary tree has the following properties:
   - the interior nodes are labeled with non-terminal symbols;
   - the leaf nodes, or the nodes along the frontier are all labeled with terminal symbols apart from one node; this node is labeled with the same non-terminal symbol as the root node and is known as the foot node; the convention outlined in [14] is followed and foot nodes are marked with an asterisk (*).

An initial tree is meant to represent a minimal non-recursive structure or derivation tree produced by the grammar, i.e. contains no recursive non-terminal symbols. Inversely, an auxiliary tree of type X represents a minimal recursive structure, which allows recursion upon the non-terminal X during derivation [17]. The set of initial trees and the set of auxiliary trees together form the set of *elementary trees*, E; where $I \cap A = \emptyset$ and $I \cup A = E$.

During derivation, composition operations are used to join elementary trees together. The adjunction operation takes an initial or derived tree a, creating a new derived tree, d by combining a with an auxiliary tree, b. A subtree, c is selected from a. The type of the subtree (the symbol at its root), X, is used to select an auxiliary tree, b, of the same type. c is removed temporarily from a. b is then attached to a as a subtree in place of c and c is attached to b by setting b's foot node as the root of c (Figure 3 depicts this idea). An example of TAG derivation is provided in the following section.

III. Tree Adjunct Grammars in Grammatical Evolution

TAGs are more powerful than CFGs which are currently used in standard GE since the set of languages produced by TAGs, TALs, is a super-set of CFLs, those produced by CFGs [14]. Unlike CFGs, however, TAGs can also generate some context-sensitive languages [7], [14]. In addition to this, it has been shown that for every CFG there is a TAG that is both weakly and strongly equivalent to it [11]. A grammar is weakly equivalent to another if it can produce the same language as the other, whereas a grammar is strongly equivalent only if it can represent each of the words in that language using the same structures as the other (i.e., derivation trees).

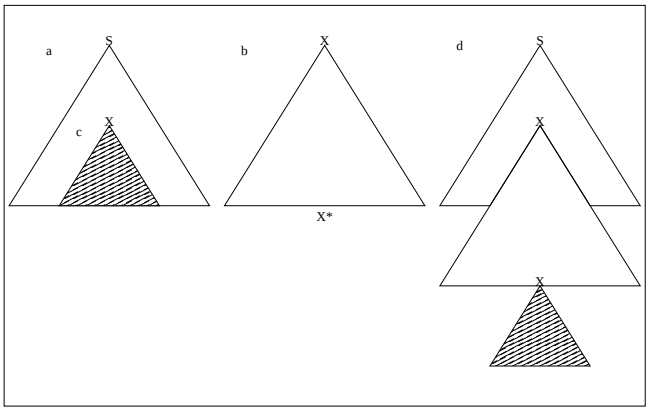


Fig. 3. Composition operation: Adjunction.

In order to incorporate the power of TAGs into GE, TAGE was developed. A number of steps were to be taken in order to achieve this. The first was to translate current CFGs used in GE into TAGs which could be used by TAGGE. Secondly, an algorithm for derivation had to be developed in order to successfully map using a TAG from genotype to phenotype.

A. CFG to TAG

There is a special type of TAG called a lexicalised TAG (LTAG). A lexicalised grammar has two defining properties:

- it consists of a finite set of structures, each with at least one terminal symbol, known as the anchor;
- it has at least one operation for composing the structures together.

The TAGs referenced in this study are LTAGs, since all the leaf nodes of the elementary trees are labeled with terminal symbols (apart from the foot nodes). The phrase LTAGs and TAGs will be inter-changable throughout the remainder of this paper and will both represent lexicalised tree-adjunct grammars.

In [14], Joshi and Schabes state that a *"finitely ambiguous CFG*[1] *which does not generate the empty string, then there is a lexicalised tree-adjunct grammar generating the same language and tree set as that CFG"*. Joshi and Schabes also provided an algorithm for generating such a TAG. This algorithm is outlined below.

Take a finitely ambiguous CFG, $G = \{NT, T, P, S\}$, where: NT is the set of non-terminal symbols; T is the set of terminal symbols; P is the set of production rules; S is the start symbol. Construct a directed graph, g, from G, where the nodes of the graph are labeled with symbols from NT and the edges of the graph are labeled with the productions from P which map between them. Then find the set of minimal cycles, c, in the graph such that they contain no other cycles within them. The productions in P must then be divided into two separate sets; R is the set of recursive productions (a production is recursive if it is part of a cycle, c_i); and NR is the set of non-recursive productions in the grammar.

Using S as the root node, create the set of all possible derivation trees using only the productions in NR. This is the set of initial trees, I. Then create A, the set of auxiliary tree, as an empty set. $A = \emptyset$.

For each node in each of the cycles, if there is a tree in $I \cup A$ which has the same label as that node, create the set of all possible derivation trees, using only the productions in NR and the current cycle, where the current node is the root node and the leaf node with the same label, as the foot node. Add this set of trees to A. For a summary of this algorithm in pseudo-code, see Algorithm 1.

Algorithm 1 Generating a TAG from a CFG

Require: $G = \{NT, T, P, S\}$
  $g = createDiGraph(G)$;
  $c = findBaseCycles(g)$;
  $R = getRecursiveProductions(P, g)$;
  $NR = P - R$;
  $I = generateInitialTrees(S, NR)$;
  $A = \emptyset$
  for all c_i in c **do**
    for all n_j in c_i **do**
      $E = I \cup A$;
      if a tree in E has a node labeled the same as n_j **then**
        $A = A \cup generateAuxiliaryTrees(n_j, c_i, NR)$
      end if
    end for
  end for

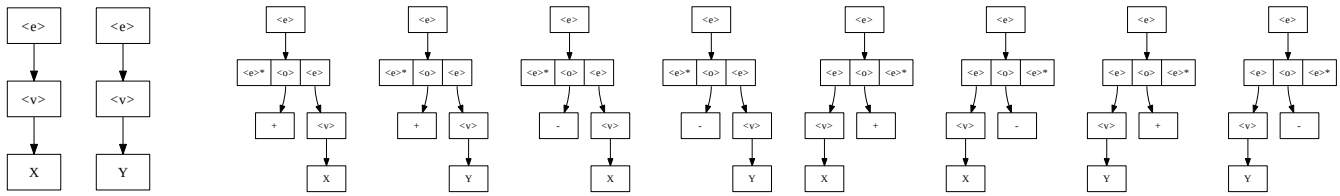
An example of a TAG produced by Algorithm 1 can be seen in Figure 4, which was generated from the CFG in Figure 1.

B. Derivation in TAGE

Derivation in TAGGE is different from derivation in GE in that it is a two step process, first a derivation tree is formed, and from that the derived tree is produced. The derivation tree is different to that of standard GE, as it is a tree where each node itself is representative of an elementary tree. The edges of the derivation tree are labeled with a node address. This address leads to a node in the tree represented by the tail of the edge. It is on this node that a composition operation is to be applied using the tree at the head of the edge. The derived tree, is the same as standard GE's derivation tree. It is the tree of symbols resulting from the composition operations listed in the derivation tree. Examples of both types of trees can be seen in Figure 5.

The derivation tree, the tree of trees, is important when dealing with TAGs if you intend to do any operations on the tree itself. For example, sub-tree crossover on the derived tree could result in altering an elementary tree, the most basic structure in a TAG. These operations should instead be used on the derivation tree, allowing whole elementary trees to be moved about.

[1]A grammar is said to be *finitely ambiguous* if all finite length sentences produced by that grammar cannot be analysed in an infinite number of ways.



(a) α_0 (b) α_1 (c) β_0 (d) β_1 (e) β_2 (f) β_3 (g) β_4 (h) β_5 (i) β_6 (j) β_7

Fig. 4. I (α) and A (β) sets of the TAG produced from the CFG in Figure 1 using the algorithm above.

1) Example Derivation: An example of derivation in TAGE follows. It is similar to the algorithm used in [8]. Given the TAG G, where $T = \{x, y, +, -\}$, $N = \{< e >, < o >, < v >\}$, $S =< e >$ and I and A are shown Figure 4, derivation proceeds using the chromosome from Figure 1 and operates as follows. First an initial tree must be chosen to start the derivation. The first codon value is read, `12`, and is used to choose an initial tree based on the number of trees in I using the same mapping function as GE, `i mod c`. From I, the set of α trees, `12 mod 2 = 0`, the zero-th tree is chosen, α_0 and set as the root node of tree `t`, the derivation tree, see Figure 5(a).

Next we enter the main stage of the algorithm. An location to perform adjunction must be chosen. The set, `N`, is created of the adjunctable addresses[2] available within all nodes(trees) contained within `t`. In this case `N` = $\{\alpha_0$`0`$\}$, so a codon is read and an address is selected from `N`, `2 mod 1 = 0` indicates which address to choose, `N`[`0`]. Adjunction will be performed at α_0`0`, or index `0` of tree α_0, `<e>`. An auxiliary tree is now chosen from the set of trees in A that are of the type `l`, i.e., the label of their root node is `l`, where `l` is the label of the node adjunction is being performed on. In this case `l = <e>`. Since there are 8 such trees in A, `3 mod 8 = 3`, β_3 is chosen. This is added to `t` as a child of the tree being adjoining to, labeling the edge with the address `0`, see Figure 5(b). The adjunctable addresses in β_3 will be added to `N` on the next pass of the algorithm. This process, the main part of the algorithm, is repeated until all remaining codons have been read. The resulting derivation and derived trees from each stage of this process can be seen in Figure 5.

If the end of the chromosome is reached and the algorithm is in the middle of execution, the individual is marked as invalid and is awarded the worst possible fitness.

IV. Experimental Setup

The focus of this study is to compare the performance of standard GE to TAGE, and to analyse the results in order to discover whether TAGs affect the ability of GE to find correct solutions.

The GEVA v1.1 software [23], [19] was used to conduct the experiments for this study. It was extended to allow the use of TAGs. The evolutionary parameters adopted for all the benchmark problems described below are presented in Table I. A short chromosome size was selected due to TAGs using the entire chromosome during derivation, unlike standard GE where the percentage of the chromosome used varies per individual. This provided TAGE the ability to attempt to find short solutions to the problems if they existed. Each run evolved for 200 generations enabling longer solutions to be explored if needed through chromosome growth by means of single-point crossover. Wrapping, as described at the end of Section II-A.1, was disabled for all experiments.

TABLE I
Grammatical Evolution parameter settings adopted for each each of the benchmark problems.

Parameter	Value
Generations	200
Population Size	100
Initialisation	Random
Initial Chromosome Size	15
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	10 Individuals
Selection Operation	Tournament
Tournament Size	3
One Point Crossover Probability	0.9
Integer Mutation Probability	0.02

A. Benchmark Problems

Standard GE was compared to TAGE using 5 classic benchmark problems from specialised GP literature. 100 independent runs were preformed for each of the problems listed below using each setup. The context free grammars used in both standard GE and used to generate TAGs for TAGE are shown in Figure 6.

Even-5-parity This problem attempts to evolve the five input even-parity boolean function, in which the best fitness is obtained when the correct output is returned for each of the 32, 2^5, test cases.

Santa Fe ant trail In this problem a control program is evolved to control the movements of an artificial ant on a toroidal grid of size 32 by 32 units. The ant has the ability to use one of several operations: **foodAhead()**, **right()**, **left()** or **move()**. The ant must use these operations to collect all 89 pieces of food located along a broken trail. **foodAhead()** enables the ant to check if there is food in the tile directly facing it. The latter three operations consume one of an

[2]An adjunctable address in a tree is the breadth first traversal index of a node labeled with a non-terminal symbol, of which there is an auxiliary tree of that type, and that there is currently no auxiliary tree already adjoined to the tree at that index

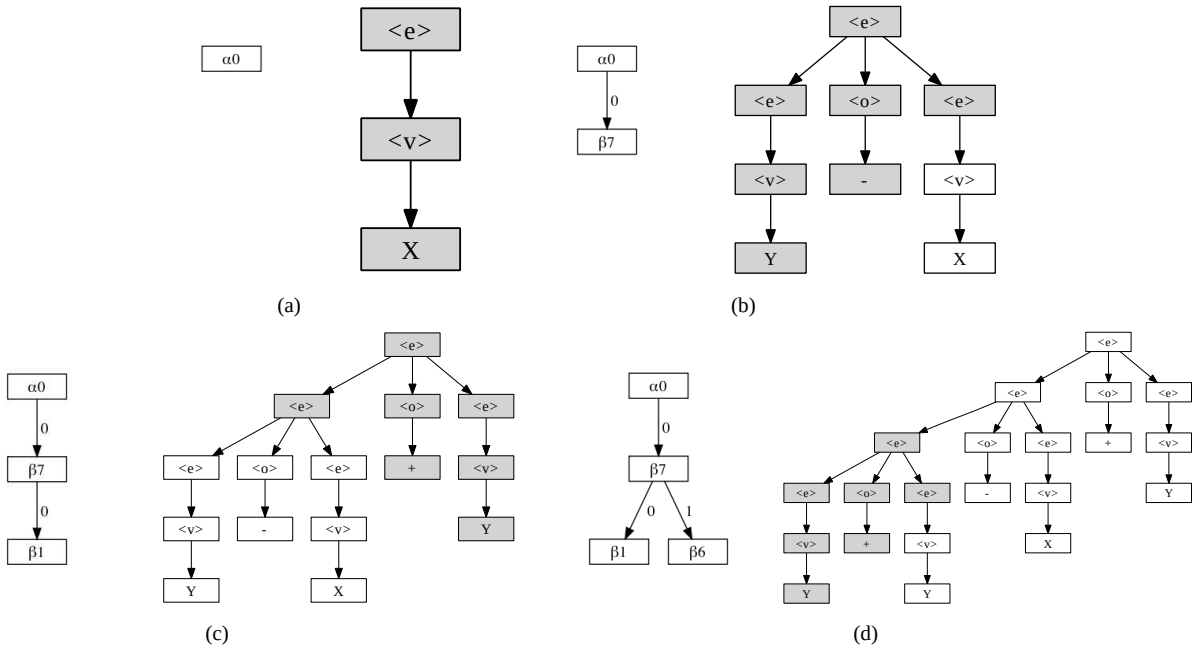


Fig. 5. The derivation tree and corresponding derived tree at each stage of derivation in TAGE. The shaded areas indicate the new content added to the tree at each step.

initial six hundred units of energy.

Symbolic Regression The objective is to evolve the classic quartic function, $x + x^2 + x^3 + x^4$. Fitness is measured by the sum of the error across 20 test cases drawn from the range $[-1, 1]$. A successful solution is where the error is less than a certain threshold, or hits criterion, as described in [16]. In this case there was a hits criterion of 0.01.

Six Multiplexer This classic GP boolean function problem in which evolution attempts to find correct two input and four output line boolean function. A perfect solution generates the correct output for a given input for all 64 test cases. Fitness is measured of how many test cases generate an incorrect output.

Max This problem, as described in [4], aims to evolve a tree whose growth is constrained by a depth limit, that when the tree's phenotype is executed, returns the largest value possible. A function set of addition and multiplication operators are used as well as a terminal set of a single value of 0.5.

V. Results and Discussion

A. Performance

The results obtained across the 100 runs for each setup are outlined in Table II. The plots for mean best fitness per generation for all the problems can be see in Figure 7. The plots for average chromosome length, average derivation tree depth and average derivation tree size per generation for the Max problem and the Even-5-parity problem (which is representative of the results found the remaining 3 problems) have also been included and can be seen in Figure 8 and Figure 9 respectively.

TABLE II

A comparison of results obtained using both standard GE and TAGE across the benchmark problems - The best and average fitness values for each population averaged across all 100 runs for the final generation, as well as the number of runs which found successful solutions to the problems.

	Best Fitness Mean (SD)	Average Fitness Mean (SD)	Successes (/100)
Even-5			
GE	2.08 (4.57)	6.15 (3.58)	79
TAGE	0.64 (1.90)	13.12 (0.67)	88
Santa Fe			
GE	32.42 (10.59)	42.13 (9.76)	3
TAGE	16.53 (10.99)	69.32 (3.96)	12
Sym.Reg.			
GE	0.253 (0.394)	8.700 (13.689)	44
TAGE	0.054 (0.171)	29.053 (9.145)	76
6 Multi.			
GE	9.14 (4.19)	11.65 (3.69)	6
TAGE	1.77 (2.66)	16.87 (1.83)	63
Max			
GE	2.31 (4.22)	199.88 (125.46)	0
TAGE	2.04 (1.52)	579.79 (335.25)	0

It is visible from both Table II and Figure 7 that for four out of the five problems examined, TAGE achieves a better mean best fitness by the final generation than standard GE, as well as having more runs which found successful solutions to those problems. TAGE fails to produce better results for the Max problem.

In addition, Figure 7 portrays that, on average, TAGE appears to have the ability to search more efficiently, finding

```
Even-5 parity grammar:
<prog>   ::= <expr>
<expr>   ::= <expr> <op> <expr>
         | ( <expr> <op> <expr> )
         | <var>
         | <pre-op> ( <var> )
<pre-op> ::= not
<op>     ::= "|" | & | ^
<var>    ::= d0 | d1 | d2 | d3 | d4
```

```
Santa Fe ant trail grammar:
<prog>      ::= <code>
<code>      ::= <line> | <code> <line>
<line>      ::= <condition> | <op>
<condition> ::= if(food\_ahead()==1)
                { <opcode> }
                else
                { <opcode> }
<op>        ::= left(); | right(); | move();
<opcode>    ::= <op> | <opcode> <op>
```

```
Max grammar:
<prog> ::= <expr>
<expr> ::= <op> <expr> <expr>
         | <var>
<op>   ::= + | *
<var>  ::= 0.5
```

```
Symbolic Regression grammar:
<expr> ::= ( <op> <expr> <expr> )
         | <var>
<op>   ::= + | - | *
<var>  ::= x0 | 1.0
```

```
Six Multiplexer grammar:
<B> ::= (<B>) &&(<B>)
      | (<B>) "||"(<B>)
      | !(<B>)
      | (<B>) ? (<B>) : (<B>)
      | a0 | a1 | d0| d1 | d2 | d3
```

Fig. 6. Grammars in Backus-Naur form used for all the benchmark problems.

better solutions in fewer generations than standard GE. This can be seen for the first four problems with the mean best fitness initially improving much more rapidly for TAGE, than GE.

One possible reason for this is that the mean average fitness values for TAGE in Table II are much larger than those for standard GE. One expects convergence to happen as the population finds a local maxima, and indeed this can be seen to be the case for standard GE where the mean average fitness values are quite close to the mean best fitness values, the difference between them being less than the sum of their standard deviations for four out of the five problems. TAGE however has mean average fitness values which are much greater than those of both standard GE's mean average fitness values and TAGE's own mean best fitness values. This might indicate that TAGE has a much lower locality than standard GE when mapping between the genotype and phenotype, i.e. a change to the genotype has a much larger effect to the phenotype in TAGE than in standard GE. Such a property causes larger jumps through the search space, promoting diversity, and as such, deters convergence.

Another possible hypothesis for why TAGE finds more successful solutions than standard GE is that, as can be seen in Figure 9, TAGE generates much deeper trees with a proportionally large amount of nodes than standard GE. Since the number of solutions with a specific fitness increases as the length of the solutions, and hence the size of the trees increase [18], it is possible that TAGE, in generating larger trees is more effective at finding these longer solutions.

B. Chromosome Length

An unforeseen result of TAGE is that the chromosome length does not grow in size uncontrollably as it does standard GE. This can be seen in Figure 9(a). One possible argument for why this happens is that the mapping process in standard GE might not use up the entire chromosome, whereas the mapping process in TAGE does. So every operation on the chromosome that affects its length has an effect on the fitness of that individual in TAGE, but in standard GE this might not always be the case. If it adversely affects the fitness of an individual in TAGE then evolution might discard this new individual, preserving the original chromosome length. In standard GE if the fitness is not affected then this individual will survive on to the next generation where the probability that a chromosome lengthening operation will not affect fitness is increased.

C. Limitations

As a result of making use of adjunction as the only composition operation, it was discovered that TAGE could not operate on complex grammars. During the translation process from CFG to TAG, the number of elementary trees grows exponentially with the complexity of the CFG, due to all possible derivations being explored, see Algorithm 1. With the introduction of substitution as a composition operation, this limitation could be overcome, since it allows for a much more compact grammar.

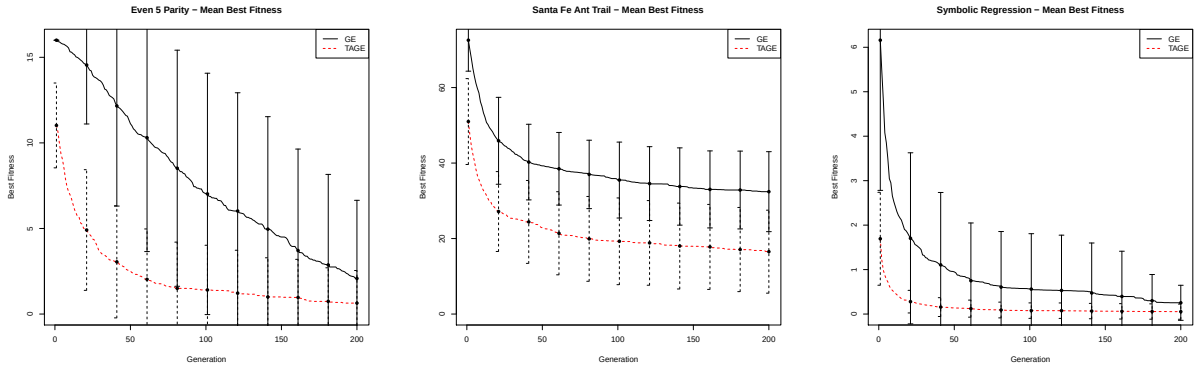
VI. Conclusions

This study presents a new form of grammatical evolution, which adopts tree-adjunct grammars in place of context-free grammars, TAGE. It demonstrates that the use of TAGs in GE has a beneficial effect on GE's ability to move through the solution search space and to find successful solutions. However, the derivation trees, and hence the solution sizes, are much larger, which is beneficial to finding solutions of a specific fitness, but not always desirable.

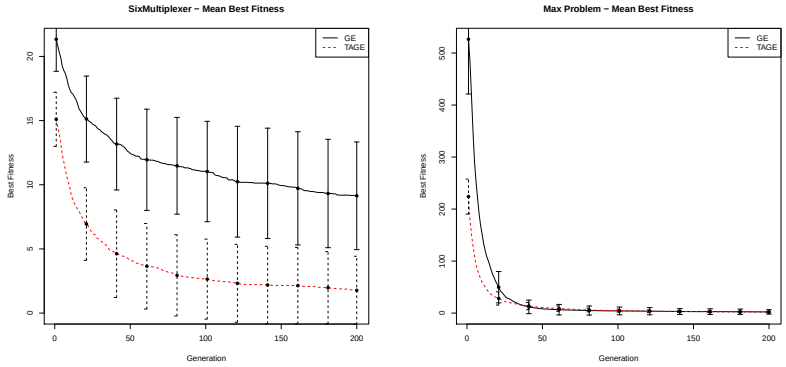
TAGE has the ability to take larger steps through the search space than standard GE, promoting diversity, while attempting to prevent convergence.

In [8], [20] TAGs were found to have a positive effect on the performance of GP, results which are shown by this study to carry over into the field of GE, with interesting results and properties of its own, such as minimal growth of the chromosome, and the apparent low locality of the mapping process between genotype and phenotype, which helps GE maintain diversity.

It is the intention of this author to continue this study by extending TAGE to work with tree-adjoining grammars and the substitution composition operation, and to investigate further the implications of using TAGs in GE and for what reasons do they perform better than CFGs in standard GE do.

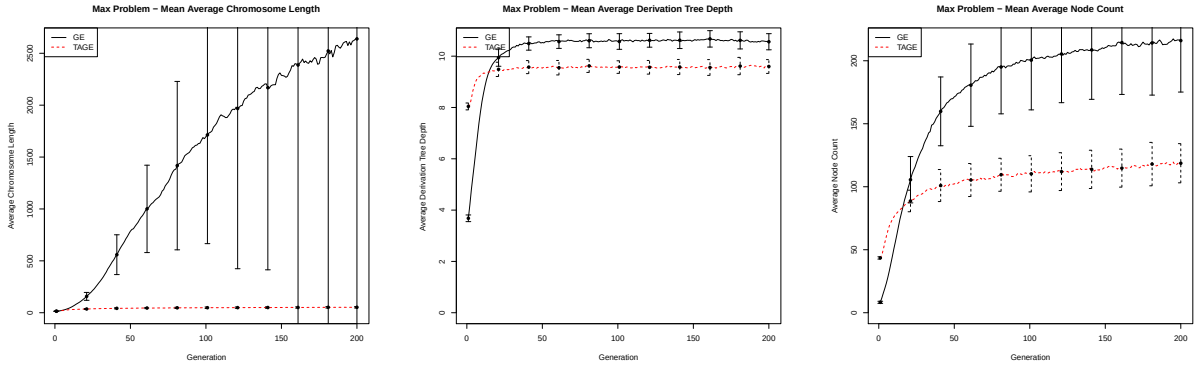


(a) Even-5 Parity (b) Santa Fe Ant Trail (c) Symbolic Regression



(d) Six Multiplexer (e) Max

Fig. 7. Mean best fitness plots across 100 runs with error bars of one standard deviation.



(a) Average chromosome length (b) Average derived tree depth (c) Average derived tree node count

Fig. 8. Result plots for the Max problem.

ACKNOWLEDGMENTS

TODO

REFERENCES

[1] R. Cleary. Extending Grammatical Evolution with Attribute Grammars: An Application to Knapsack Problems, 2005.

[2] J. J. Collins, C. Ryan, and M. O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. *Lecture Notes in Computer Science*, 1391:8396, 1998.

[3] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments, 1st edition*. 2009.

[4] C. Gathercole and P. Ross. An adverse interaction between crossover and restricted tree depth in genetic programming. *Genetic And Evolutionary Computation Conference*, 1996.

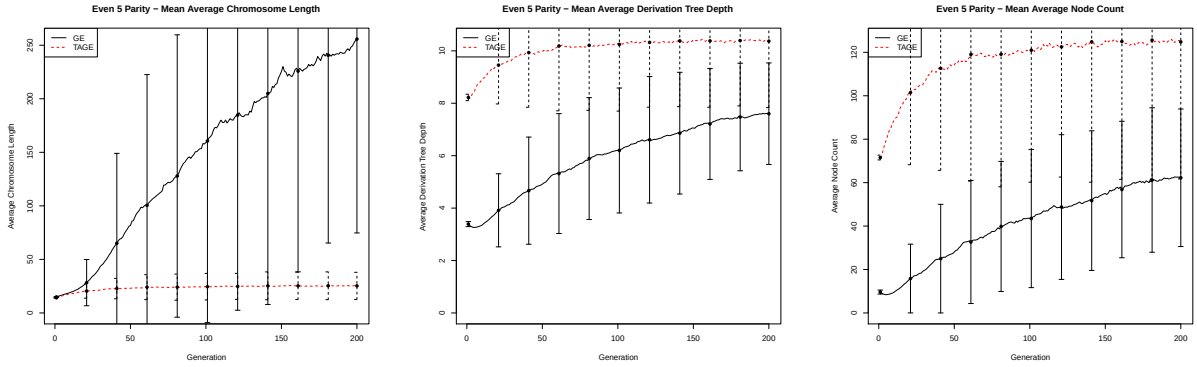
[5] M. Hemberg and U. OReilly. Extending Grammatical Evolution to Evolve Digital Surfaces with Genr8, 2004.

[6] N. Hoai. A Flexible Representation for Genetic Programming: Lessons from Natural Language Processing, 2004.

[7] N. Hoai and R. McKay. A framework for tree adjunct grammar guided genetic programming. pages 93–99. ADFA, 2001.

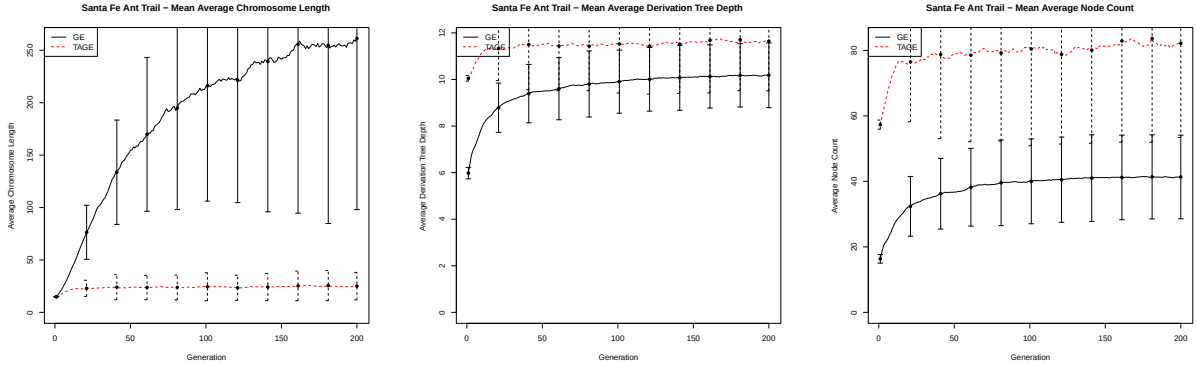
[8] N. Hoai, R. McKay, and D. Essam. Some experimental results with tree adjunct grammar guided genetic programming. *Lecture notes in computer science*, page 228237, 2002.

[9] N. Hoai, R. McKay, and D. Essam. Representation and structural difficulty in genetic programming. *IEEE Transactions on Evolutionary*



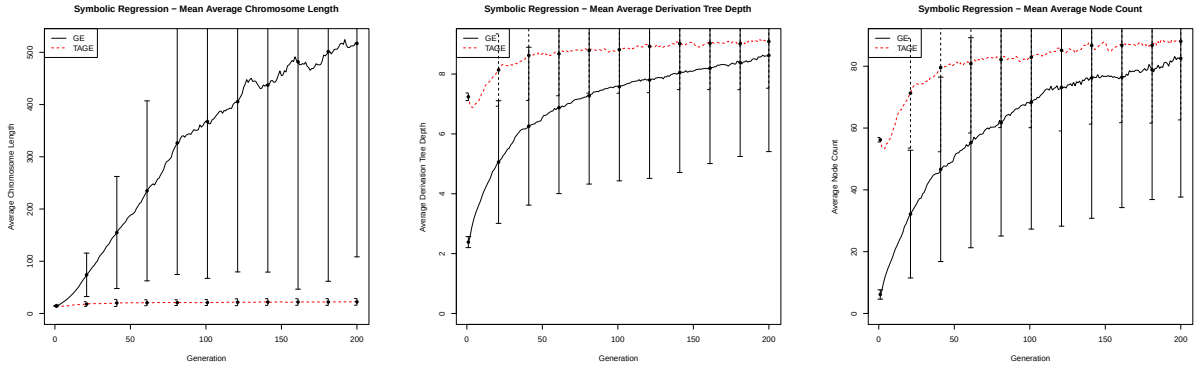
(a) Average chromosome length (b) Average derived tree depth (c) Average derived tree node count

Fig. 9. Result plots for the Even Five Parity problem. Similar trends are found in all other problems with the exception of the Max problem.



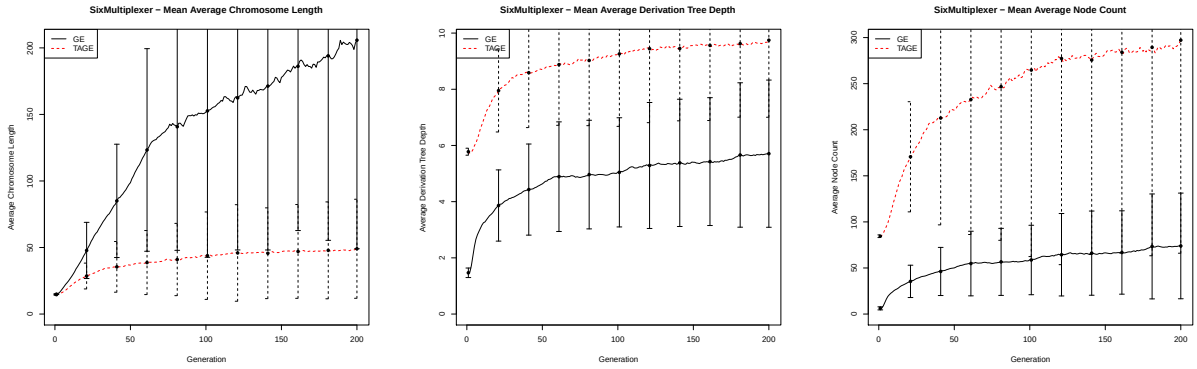
(a) Average chromosome length (b) Average derived tree depth (c) Average derived tree node count

Fig. 10. SF



(a) Average chromosome length (b) Average derived tree depth (c) Average derived tree node count

Fig. 11. SR



(a) Average chromosome length (b) Average derived tree depth (c) Average derived tree node count

Fig. 12. 6multi

Computation, 10(2):157–166, April 2006.

[10] N. Hoai, R. McKay, D. Essam, and R. Chau. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, pages 1326–1331, 2002.

[11] A. Joshi. *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions*, chapter 6, pages 205–250. Cambridge University Press, New York, 1985.

[12] A. Joshi. *An Introduction to Tree Adjoining Grammars*. John Benjamins, Amsterdam, 1987.

[13] A. Joshi, L. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163, 1975.

[14] A. Joshi and Y. Schabes. Tree-Adjoining Grammars. *Handbook of Formal Languages, Beyond Words*, 3:69123, 1997.

[15] M. Keijzer, C. Ryan, M. O'Neill, M. Cattolico, and V. Babovic. Adaptive Logic Programming. In *Genetic and Evolutionary Computation Conference*, 2001.

[16] J. Koza. *Genetic Programming*. MIT Press, 1992.

[17] A. Kroch and A. Joshi. The Linguistic Relevance of Tree Adjoining Grammar, 1985.

[18] W. Langdon, T. Soule, R. Poli, and J. Foster. The Evolution of Size and Shape. In L. Spector, W. Langdon, U. O'Reilly, and P. Angeline, editors, *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, Cambridge, MA, USA, June 1999.

[19] J. McDermott, M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, and A. Brabazon. GEVA: grammatical evolution in Java, 2008.

[20] R. McKay, N. Hoai, and H. Abbass. Tree adjoining grammars, language bias, and genetic programming. *Lecture notes in computer science*, page 335344, 2003.

[21] M. O'Neill, A. Brabazon, M. Nicolau, S. Garraghy, and P. Keenan. piGrammatical Evolution, 2004.

[22] M. O'Neill, D. Fagan, E. Galvan, A. Brabazon, and S. McGarraghy. An analysis of Genotype-Phenotype Maps in Grammatical Evolution. In *EuroGP 2010*, 2010.

[23] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott, and A. Brabazon. GEVA - Grammatical Evolution in Java, 2008.

[24] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. 2003.

[25] M. O'Neill, J. Swafford, J. McDermott, J. Byrne, A. Brabazon, E. Shotton, C. McNally, and M. Hemberg. Shape grammars and grammatical evolution for evolutionary design. In *Genetic And Evolutionary Computation Conference*, 2009.

[26] R. Poli, W. Langdon, and N. McPhee. *A Field Guide To Genetic Programming*. lulu, 2008.