A Comparison of GE and TAGE in Dynamic Environments

Eoin Murphy Michael O'Neill
Anthony Brabazon

Natural Computing Research and Applications Group,
Univeristy College Dublin, Ireland.
{eoin.murphy,m.oneill,anthony.brabazon}@ucd.ie

Abstract

The lack of study of genetic programming in dynamic environments is recognised as a known issue in the field of genetic programming. This study compares the performance of two forms of genetic programming, grammatical evolution and a variation of grammatical evolution which uses tree-adjunct grammars, on a series of dynamic problems. Mean best fitness plots for the two representations are analysed and compared.

1 Introduction

Genetic Programming (GP) research has most commonly been applied to static or toy problems, since the properties of these problems are well understood. This helps researchers identify the effects of their research when attempting to solve problems. Applying GP to real world problems in dynamic and varying environments is much harder since the problem domain is not as well understood. This can make it more difficult to comprehend the effects of the research. It is not clear if improvements discovered while searching static environments cross over when applied to dynamic problems. Indeed, dynamic environments has been recognised as an open issue for investigation in GP [17].

Grammatical Evolution (GE), a grammar-based form of GP[3, 12, 16], which traditionally uses context-free grammars (CFG), was extended to use tree-adjunct grammars (TAG) [6, 5] in the form of Tree-Adjunct Grammatical Evolution (TAGE) [15]. TAGE showed promising improvements in performance when applied to a series of static problems. Improvements such as finding better solutions in fewer generations and finding more perfect solutions than traditional GE on those problems [15].

Subsequently, it has been shown by Murphy et al. [14] that the TAGE mutation landscapes have much greater connectivity than those of GE when a subset of the above problems were examined. It was noted that this could be partially responsible for TAGE's improved performance in search [14].

In this study we investigate if TAGE provides an advantage over GE on a series of dynamic problems of varying dynamism.

The remainder of this study is laid out as follows: descriptions of GE, TAGE and DEs in Section 2; the experimental work is outlined in Section 3, with the results and discussion presented in Sections 4 and 5; finally, conclusions and future work are outlined in Section 6.

2 Background information

This section provides a brief introduction into GE and TAGE, as well as a definition the types of DEs that are used for this study.

```
Grammar:
<e>:= <e><o><e> | <v>
<o>:= + | -
<v>:= x | y

Chromosome:
12, 3, 7, 15, 9, 36, 14
```

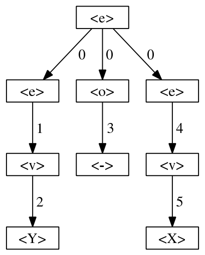


Figure 1: Example GE grammar, chromosome and resulting derivation tree.

2.1 Grammatical Evolution

GE is a grammar-based approach to GP, combining aspects of Darwinian natural selection, genetics and molecular biology with the representational power of grammar formalisms [3, 12, 16]. The grammar, written in Backus-Naur form, enables GE to define and modify the legal expressions of an arbitrary computer language. Moreover, the grammar also enables GE to modify the structure of the expressions generated, something that is not trivial for other forms of GP. In addition, the separation of the genotype from the phenotype in GE allows genetic operations to be applied not only to the phenotype, as in GP, but also to the genotype, extending the search capabilities of GP. GE is considered to be one of the most widely applied GP methods today [12].

2.1.1 GE Derivation Example

Representation in GE consists of a grammar and a chromosome, see Fig. 1. A genotype-phenotype mapping uses values of codons in the chromosome to select production rules from the grammar, building up a derivation tree. The phenotype string can be extracted from this leaf nodes of this derivation tree.

The mapping begins with the start symbol, `<e>`. The value of the first codon, `12`, is read from the chromosome. The number of production rules for the start symbol are counted, `2`, `<e><o><e>` and `<v>`. The rule to be chosen is decided according to the mapping function `i mod c`, where `i` is the current codon value and `c` is the number of choices available, e.g, `12 mod 2 = 0`, therefore the zero-th rule is chosen. `<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 children. The next symbol to expand is the first non-terminal leaf node discovered while traversing the tree in a depth first manner. In this case the left-most `<e>` is chosen. The next codon, `3`, is read, expanding this `<e>` to `<v>` and growing the tree further. The next symbol, `<v>` is expanded using the next codon, `7`. `7 mod 2 = 1`, so the rule at index 1, `Y`, is chosen.

Derivation continues until there are no more non-terminal leaf nodes to expand, or until the end of the chromosome has been reached. If there are non-terminal leaf nodes left when the end of the chromosome has been reached, derivation can proceed in one of a few different manners. For example, a bad fitness can be assigned to the individual, so it is highly unlikely that this individual will survive the selection process. Alternatively the chromosome can be wrapped, reusing it a predefined number of times. If after the wrapping limit has been reached and the individual is still invalid, it could then be assigned a bad fitness. The complete derivation tree for this example is shown in Fig. 1.

2.2 Tree-Adjunct Grammatical Evolution

TAGE, like GE, uses a representation consisting of a grammar and a chromosome. The type of grammar used in this case is a TAG rather than a CFG. A TAG is defined by a quintuple (T, N, S, I, A) where *a)* T is a finite set of terminal symbols; *b)* N is a finite set of non-terminal symbols: $T \cap N = \emptyset$; *c)* S is the start symbol: $S \in N$; *d)* I is a finite set of finite trees called *initial trees* (or α trees); *e)* A is a finite set of finite trees called *auxiliary trees* (or β trees). Initial trees have the following properties: their root nodes are labeled with S and the interior nodes are labeled with non-terminal symbols. An initial tree's leaf nodes

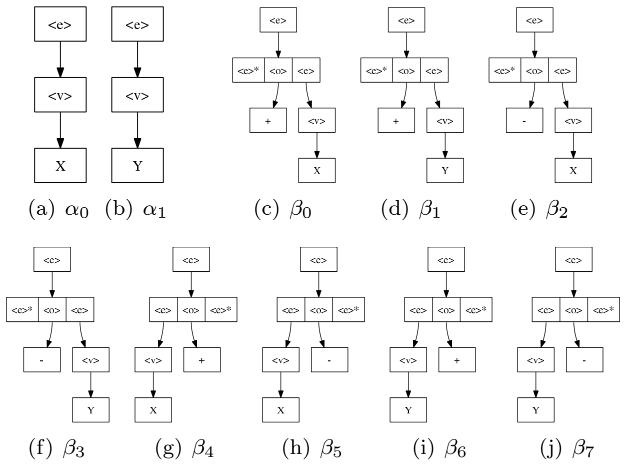


Figure 2: Initial and auxiliary tree sets of the TAG produced from the CFG in Fig. 1.

are labeled with terminal symbols. The interior nodes of auxiliary trees are also labeled with non-terminal symbols, as well as their leaf nodes being labeled with terminal symbols. However, one special leaf node, the foot node, is labeled with the same non-terminal symbol as the root. Foot nodes are marked with * [6].

An initial tree represents a minimal non-recursive structure produced by the grammar, i.e., it 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 [9]. The union of initial trees and auxiliary trees forms the set of *elementary trees*, E; where $I \cap A = \emptyset$ and $I \cup A = E$.

During derivation, composition operations 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 sub-tree, c is selected from a. The type of the sub-tree (the symbol at its root) 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 sub-tree in place of c and c is attached to b by replacing c's root node with b's foot node. An example of TAG derivation is provided in Section 2.2.1.

2.2.1 TAGE Derivation Example

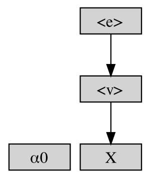
TAGE generates TAGs from the CFGs used by GE. Joshi and Schabes [6] state that for a *"finitely ambiguous CFG*[1] *which does not generate the empty string, there is a lexicalised tree-adjunct grammar generating the same language and tree set as that CFG"*. An algorithm was provided by Joshi and Schabes [6] for generating such a TAG. The TAG produced from Fig. 1 is shown in Fig. 2.

Derivation in TAGE is different to GE. A TAGE derivation tree is a tree of trees. That is to say, a node in a TAGE derivation tree contains an elementary tree. The edges between those nodes are labeled with a node address of the tree in the parent derivation node. It is at this address that the auxiliary tree in the child node is to be adjuncted. A derived tree in TAGE is a tree of symbols, similar to GE's derivation tree, resulting from the application of the adjunction operations defined in the TAGE derivation tree.

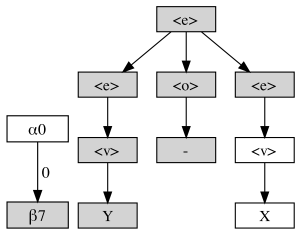
Given the TAG G, where $T = \{x, y, +, -\}$, $N = \{< e >, < o >, < v >\}$, $S =< e >$ and I and A are shown Fig. 2, derivation, using the chromosome from Fig. 1, operates as follows. An initial tree is chosen to start derivation. The first codon value, `12`, is read and is used to choose an initial tree based on the number of trees in I. Using the same mapping function as GE, `12 mod 2 = 0`, the zero-th tree, α_0, is chosen from I. This tree is set as the root node of, `t`, the derivation tree, see Fig. 3(a).

Next we enter the main stage of the algorithm. A location to perform adjunction must be chosen. The set `N` is created of the adjunct-able addresses available within all nodes(trees) contained within `t`. An adjunct-

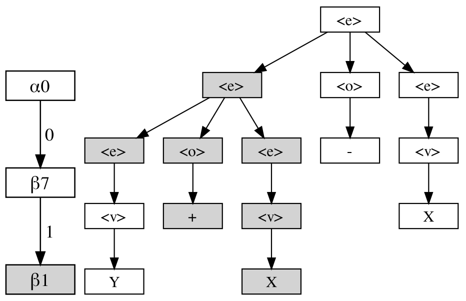
[1]A grammar is said to be *finitely ambiguous* if all finite length sentences produced by that grammar cannot be analysed in an infinite number of ways.



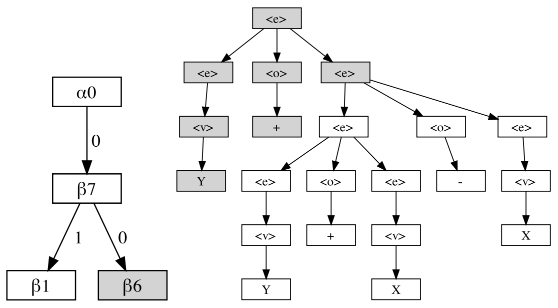
(a) The initial tree α_0.



(b) β_7 adjoined to α_0 at address 0.



(c) β_1 adjoined to β_7 at address 1.



(d) β_6 adjoined to β_7 at address 0.

Figure 3: 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.

able 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 there is currently no auxiliary tree already adjoined to the tree at that index. In this case `N` = $\{\alpha_0 0\}$, so a codon is read and an address is selected from `N`, `3 mod 1 = 0` indicating which address to choose, `N[0]`. Adjunction will be performed at $\alpha_0 0$, or index `0` of tree α_0, `<e>`. An auxiliary tree is now chosen from A that is of the type `l`, i.e., the label of it's root node is `l`, where `l` is the label of the node adjunction is being performed at. In this case `l = <e>`. Since there are 8 such trees in A, `7 mod 8 = 7`, β_7 is chosen. This is added to `t` as a child of the tree being adjoining to, labeling the edge with the address `0`, see Fig. 3(b). The adjunct-able addresses in β_7 will be added to `N` on the next pass of the algorithm. This process is repeated until all remaining codons have been read. The resulting derivation and derived trees at each stage of this process can be seen in Fig. 3.

2.3 Dynamic Environments

It has been shown that dynamic environments can help speed up synthetic evolution in certain cases [7]. Additionaly, it has been theorised that the dynamism of the natural world has helped natural organisms evolve into the complex systems that they are today [7, 18, 19]. As such it is important to study dynamic environments in an attempt to understand their properties on natural evolution and how these properties can be exploited when evolving in silico.

In order to study dynamic environments it is important to define what exactly is meant by *dynamic*. There are many ways in which evolutionary algorithms can be classed as dynamic [3, 13, 2]. In the context of genetic programming, the constraints of a problem, the inputs, or even the objective function itself could be changed with respect to time. Defining exactly which of these types of change is *dynamic* is difficult as it can be problem specific. This paper addresses the definition of dynamism where there is a functional change over time, i.e. the objective function, and thus the fitness landscape, change with respect to time. By that definition the objective of evolution depends on time t:

$$f(x) := f(x, t) \tag{1}$$

where f is the fitness function, x is an individual and t is the current generation. The majority of problems in this study follow this definition but one problem examined changes the problem constraints with respect to time and is described in section 3.1.

Two important properties of dynamic environments are *a*) the frequency of change, how many generations between changing environments and *b*) the magnitude of change, how much the environment/fitness landscape changes [20]. This study examines a number of problems each with different magnitudes of change, across a range of different frequencies. The goal of this study is to discover the advantages and disadvantages of using the TAGE representation over the standard GE representation when evolving in such environments.

3 Experimental Work

This section outlines the different experiments conducted by this study in order to compare TAGE with GE when searching in dynamic environments. A series of different problems from the static domain were taken and extended to operate as dynamic problems, each with a varying range of dynamism. These problems are outlined in section 3.1 below.

In order to achieve dynamism the GE algorithm was extended to allow the fitness function to change with respect to time. The population is allowed to evolve normally for a certain number of generations before the environment changes. This number of generations is know as T, the period, and it is inversely proportional to the frequency of change, $F = 1/T$. Once the end of a period is reached, the entire population, including the elites, are re-evaluated on the new environment. Evolution then contines for T more generations before the environment changes once more.

3.1 Dynamic Problems

A number of well understood problems from the static domain have been extended into the dynamic domain, enabling the form of the problem to change over time. In terms of GP, this is implemented by extending the fitness function to allow the target solution to be modified after a certain number of generations has past. Each of the problems are described below with fitness is being minimised for all problems.

Symbolic Regression The static version of this problem tries to find the expression $f(x) = 1 + x + x^2 + x^3 + x^4$. The fitness is calculated as the sum of the error between the evolved expression and the target expression when tested on a range of inputs (20 samples between -1 and 1). The dynamic version used for this study allows each operator to vary between $+$ and $-$. Starting from all $+$ operators as seen above, the operators are changed to $-$ from left to right, interpolating between the original and final expression, $-1 - x - x^2 - x^3 - x^4$. This was chosen due to the small magnitude of genotypic change between each neighbouring expression in the series. For both representations, with the ideal chromosome each expression is only one mutation distant from it's neighbouring expressions in the series.

N-Multiplexer The classic GP input/output line boolean function. Fitness is measured by how many of the test cases generate correct outputs. The problem was extended so that different values for N could be chosen, $N \in \{3, 6, 11\}$ at each period.

Even N Parity The N input even-parity boolean function, in which the best fitness is obtained when the correct output is returned for each of the 2^N test cases. A value for $N \in \{3, 4, 5\}$ is chosen at each period and the population is evaluated against that form of the problem for T generations.

Dynamic Ant This is a newly formed problem [4]. It is a variation on the classic ant trail problems such as Santa Fe [8, 11]. The problem builds upon work by Langdon and Poli [10]. The aim is to evolve an

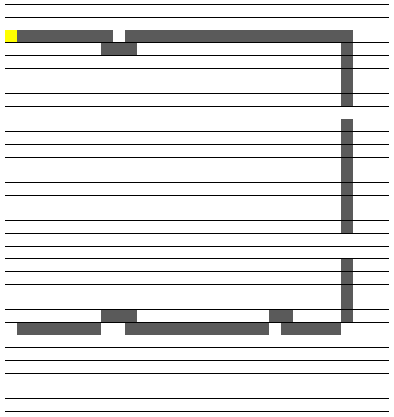


Figure 4: The dynamic ant trail - The ant starts from the top left cell in the trail.

ant controller that successfully eats the maximum number of food pellets possible. However, the ant's energy constraint changes with respect to time and the trail is engineered so that the ant must behave differently at each of the different energy levels in order to maximise the food eaten. The trail used, which is different from the classic Santa Fe ant trail, can be seen in Fig. 4.

Five different energy levels are used in the problem $20, 42, 60, 100$, and 140. At the start of each period a new energy level is chosen. Each successive energy level allows the ant to progress further along the trail, changing the maximum number of food pellets that can be eaten. Each section of the trail presents a different challenge for the ant, so in order to eat the maximum number of pellets the ant's behaviour must change. However, the optimal behaviour for one energy level will not allow the ant to eat the max number of pellets at other energy levels.

The first energy level encourages the ant to ignore turning, since the energy wasted turning to move around the small outcrop in the trail would result in not gathering as much food as continuing straight and crossing the gap. The second energy level causes the ant to learn how to handle corners, and as such can no longer just move straight. The third introduces gaps with no food to guide the ant, the fourth is to navigate the corners with gaps and the final energy level is to enable the ant to collect all the food pellets in the trail.

For the problems described above, if different instances of the problem use different numbers of input variables, the grammar for the largest number of input variables is used. For the instances of the problem which don't make use of all input variables, unused inputs are set to 0 or false, i.e., the 11-multiplexer grammar is used for all three instances of the multiplexer problem, with unused inputs set to 0 for the lower values of N. The grammars used for each of the problems can be seen in Fig. 5.

3.2 Experimental Settings

The evolutionary parameters used on each of the problems can be seen in Table 1. 100 independent runs were performed for the symbolic regression and dynamic ant problems, with 15 independent runs being performed for the remaining two problems. The random number generator was seeded the same for the GE runs as the corresponding TAGE runs.

In order to have a more complete view of the spectrum of dynamic environments two different setups were used:

1. **Incremental cyclic** where the problem begins from its simplest least complex form, and increments in complexity at each period before cycling back to it's initial form and repeating the process until the maximum number of generations has been reached;

```
Even-N 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

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

```
Dynamic Ant grammar:
<code> ::= <code> <line> | <line>
<line> ::= if(food_ahead() == 1) {
              <code>
           } else {
              <code>
           }
         | <op>
<op>   ::= left(); | right(); | move();

N Multiplexer grammar:
<B> ::= (<B>)&&(<B>)
      | (<B>)"||"(<B>)
      | !(<B>)
      | (<B>) ? (<B>) : (<B>)
      | a0 | a1 | a2 | d0| d1 | d2 | d3 | d4| d5 | d6 | d7
```

Figure 5: CFG grammars in Backus-Naur form used for all the dynamic problems.

Table 1: GE parameters adopted for each of the problems.

Parameter	**Value**
Generations	200
Population Size	500
Initialisation	Random
Initial Chromosome Size	15
Max Chromosome Wraps	0
Replacement Strategy	Generational
Elitism	10%
Selection Operation	Tournament
Tournament Size	1%
One Point Crossover Probability	0.9
Integer Mutation Probability	0.02

2. **Random cyclic** where each independent run performed generates a random permutation of the above cycle of problem states. The same permutation is generated for both GE and TAGE. This permutation is iterated through at each period, cycling until the generations run out.

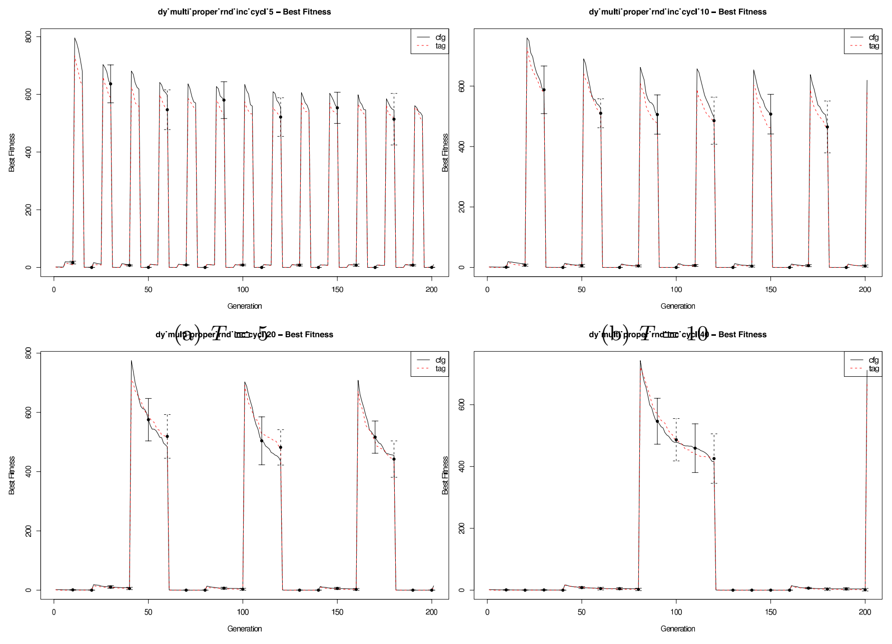
Each of the above setups were used with 4 different period lengths, $T \in \{5, 10, 20, 40\}$, i.e. different frequencies of change. In this study we wish to determine if TAGE or GE has a performance advantage on dynamic problems.

4 Results

The results of the experiments are listed in this section. The mean (across the runs) best fitness plots were generated and a sample of them can be seen in Fig. 6 and Fig. 7. Three properties of these graphs are examined (see Table 4):

Area under the curve (AUC) is calculated for each line in the mean best fitness plot. The TAGE AUC is taken as a ratio of the GE AUC. Since fitness is being minimised, a lower ratio means the TAGE is performing better on average across the run.

Fall Off (FO) is the immediate difference in mean best fitness when a change in the environment occurs, i.e., the fitness differential between the final generation of a period and the first generation of the next period. The mean FO across the entire plot is calculated and the distance between TAGE's mean FO



(a) $T = 5$ (b) $T = 10$

(c) $T = 20$ (d) $T = 40$

Figure 6: N Multiplexer Incremental Cyclic - Best fitness plots. T is the period length in generations.

and GE's mean FO is taken as a ratio against GE's mean FO. The sign of this value determines which representation recovers better, on average, when a change in the environment occurs. As fitness is being minimised a negative value indicates TAGE performs better, whereas positive means GE did. The magnitude is the ratio of the difference in mean FO's against GE's mean FO.

Drawdown (DD) describes the fitness differential from the start of a period to the end of it. This value is averaged across all of the periods and a ratio of TAGE's DD is taken against GE's DD. If the value is > 1.0 GE's fitness has improved more than TAGE's. Whereas if the value is < 1.0, then TAGE has a greater fitness differential across the period than GE.

5 Discussion

Interpreting the plots seen in Fig. 6 and Fig. 7 visually is difficult. In particular when the there are large magnitudes of change happening in the environment as can be seen in Fig. 6 when the value for N changes from 6 to 11. As a result, values on the plot were compiled in to the statistics seen in Table 4.

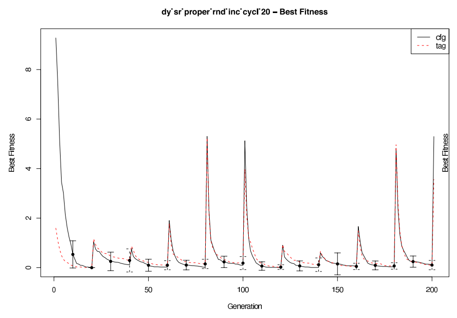
From the values for AUC in Table 4 it can be seen that on average TAGE's mean best fitness plot tends to be slightly lower than that of GE (values < 1.0). Since these plots are minimising fitness, this corresponds to TAGE populations being slightly fitter across the dynamic runs. Interestingly this trend is more evident in the random cyclic set up, this could be an indicator that GE is better able to exploit the incremental nature of the first setup than TAGE. This effect is seen to a lesser extent in the even n parity and n multiplexer problems and can most likely be attributed to the difficulty of some of the more complex forms of those problems, and the percentage of the AUC attributed to those forms, e.g. 11 bit multiplexer.

In addition to this, the large number of negative FO values in the table may indicate that TAGE populations don't converge as much as GE populations. This might help maintain more solutions with better future fitness within the population than GE does for when the environment changes. This shows that TAGE may help maintain population memory better than GE.

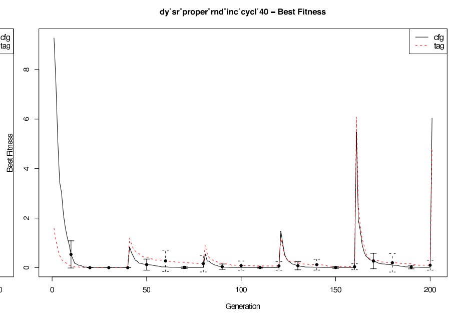
Table 4 also shows that on average over a period, GE populations have a larger fitness differential than TAGE populations (DD values < 1.0). This in a fact is not surprising, since according to the AUC and

Table 2: The AUC (and the t-test p-value of GE against TAGE), FO and DD for each of the problems across the two setups for a variety of frequencies of change. Values for AUC < 1.0 indicate TAGE's best fitness is on average, better than that of GE. The value is a ratio against GE. Positive FO values indicate TAGE's best fitness increases more than GE's does on average, a negative value indicates the opposite. The magnitude of the value is the ratio of the differential between TAGE and GE's FO values with respect to GE's value. DD is the ratio of TAGE's average fitness differential across a period against the same for GE. Values < 1.0 indicate on average a greater fitness differential across a period for GE.

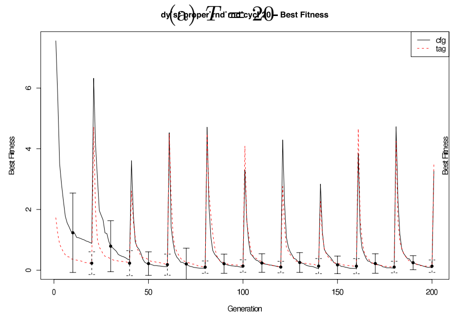
	Inc. 5	**Inc. 10**	**Inc.20**	**Inc. 40**	**Rnd. 5**	**Rnd. 10**	**Rnd. 20**	**Rnd. 40**
Sym. Reg.								
AUC	1.22	0.88	0.81	0.84	0.83	0.76	0.76	0.52
AUC p-value	0.034	0.084	0.004	0.035	0.026	0.001	0.003	0.00
FO	0.07	0.06	-0.12	-0.04	-0.12	-0.08	-0.08	-0.32
DD	0.96	0.89	0.70	0.60	0.82	0.82	0.79	0.52
Dynamic Ant								
AUC	0.89	0.93	0.95	0.97	0.88	0.91	0.94	0.94
AUC p-value	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
FO	-0.65	-0.65	-0.56	-0.50	-0.48	-0.44	-0.52	-0.44
DD	0.35	0.35	0.43	0.50	0.62	0.66	0.63	0.71
Even N Parity								
AUC	0.98	0.99	0.98	0.99	0.97	0.97	0.94	0.95
AUC p-value	0.109	0.811	0.333	0.474	0.004	0.268	0.007	0.168
FO	-0.19	-0.14	-0.26	-0.15	-2.80	-1.41	-0.50	-0.71
DD	0.51	0.48	0.50	0.45	0.52	0.45	0.70	0.66
N Multiplexer								
AUC	0.93	0.93	1.00	1.00	1.00	0.94	0.98	0.95
AUC p-value	0.234	0.353	0.732	0.978	0.491	0.872	0.567	0.619
FO	-0.11	-0.14	-0.22	-0.07	0.37	0.03	-0.24	-0.13
DD	0.88	0.81	0.78	0.91	1.15	0.99	0.74	0.87



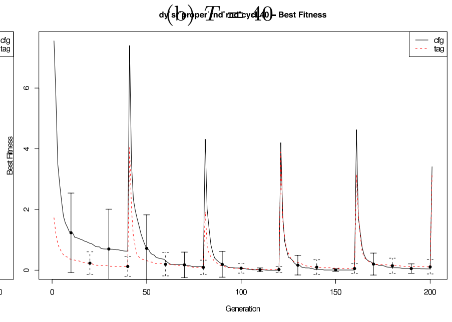
(a) $T = 20$



(b) $T = 40$



(c) $T = 20$



(d) $T = 40$

Figure 7: Symbolic Regression Incremental Cyclic *a*) and *b*) and Random Cyclic *c*) and *d*) - Best fitness plots. T is the period length in generations.

FO values GE populations are slightly less fit than TAGE populations as well as being less fit after the environment has changed, so while TAGE may be stuck in a local optima GE has not even reached this level of fitness and can try to catch up before the environment changes again.

6 Conclusions

The aim of this study was to investigate the effectiveness of two different forms of GP in dynamic environments, a topic which is recognised as an open issue in the field of GP [17]. In particular this study compares the use of CFGs against the use of TAGs with GE on a series of dynamic problems.

For the problems and grammars examined across the different setups, it was shown that TAGE on average performs slightly better than GE. However, that on problems of high difficulty, or dynamic environments with high magnitude of change, there seems to be no advantage. It was also shown that the TAGE representation may help maintain a greater population memory than GE.

One clear result of this study is the indication that there is a need for the development of benchmark dynamic problems for the field of GP, similar to the moving peaks problem in other evolutionary algorithm fields [1].

Future work emerging from this study is to investigate creating better benchmark dynamic problems for the field of GP/GE, as well as to further study the advantages and disadvantages of using the GE and TAGE representations in dynamic environments

Acknowledgments

This research is based upon works supported by the Science Foundation Ireland under Grant No. 08/IN.1/I1868.

References

[1] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages 3 vol. (xxxvii+2348), 1999.

[2] J. Branke. *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[3] I. Dempsey, M. O'Neill, and A. Brabazon. *Foundations in Grammatical Evolution for Dynamic Environments.* Studies in Computational Intelligence. Springer, 2009.

[4] D. Fagan, M. Nicolau, E. Hemberg, M. O'Neill, and A. Brabazon. Dynamic ant: Introducing a new benchmark for genetic programming in dynamic environments. In *Proceedings of the 13th Annual conference on Genetic and evolutionary computation*, GECCO '11, New York, NY, USA, 2011. ACM.

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

[6] A. Joshi and Y. Schabes. Tree-Adjoining Grammars. *Handbook of Formal Languages, Beyond Words*, 3:69–123, 1997.

[7] N. Kashtan, E. Noor, and U. Alon. Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716, 2007.

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

[9] A. Kroch and A. Joshi. The Linguistic Relevance of Tree Adjoining Grammar, Technical Report, University Of Pennsylvania, 1985.

[10] W. B. Langdon and R. Poli. Genetic programming bloat with dynamic fitness. In W. Banzhaf, R. Poli, M. Schoenauer, and T. Fogarty, editors, *Genetic Programming*, volume 1391 of *Lecture Notes in Computer Science*, pages 97–. Springer Berlin / Heidelberg, 1998.

[11] W. B. Langdon and R. Poli. Why ants are hard. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 193–201, University of Wisconsin, Madison, Wisconsin, USA, 22-25 July 1998. Morgan Kaufmann.

[12] R. McKay, N. Hoai, P. Whigham, Y. Shan, and M. O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11:365–396, 2010.

[13] R. W. Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. SpringerVerlag, 2004.

[14] E. Murphy, M. O'Neill, and A. Brabazon. Examining mutation landscapes in grammar based genetic programming. In *Proc. of the 14th European Conference on Genetic Programming, EuroGP 2011*, volume 6621 of *LNCS*, pages 131–142, Turin, Italy, 27-29 Apr. 2011. Springer Verlag.

[15] E. Murphy, M. O'Neill, E. Galvan-Lopez, and A. Brabazon. Tree-adjunct grammatical evolution. In *2010 IEEE World Congress on Computational Intelligence*, pages 4449–4456, Barcelona, Spain, 18-23 July 2010. IEEE Computational Intelligence Society, IEEE Press.

[16] M. O'Neill and C. Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*, volume 4 of *Genetic programming*. Kluwer Academic Publishers, 2003.

[17] M. O'Neill, L. Vanneschi, S. Gustafson, and W. Banzhaf. Open issues in genetic programming. *Genetic Programming and Evolvable Machines*, 11(3/4):339–363, Sept. 2010. Tenth Anniversary Issue: Progress in Genetic Programming and Evolvable Machines.

[18] M. Parter, N. Kashtan, and U. Alon. Environmental variability and modularity of bacterial metabolic networks. *BMC Evolutionary Biology*, 7(1):169, 2007.

[19] M. Parter, N. Kashtan, and U. Alon. Facilitated variation: How evolution learns from past environments to generalize to new environments. *PLoS Comput Biol*, 4(11):e1000206, 11 2008.

[20] P. Rohlfshagen, P. K. Lehre, and X. Yao. Dynamic evolutionary optimisation: an analysis of frequency and magnitude of change. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1713–1720, New York, NY, USA, 2009. ACM.