Subtree Deactivation Control with Grammatical Genetic Programming in Dynamic Environments

Michael O'Neill, Anthony Brabazon and Erik Hemberg

Abstract— **We investigate the usefulness of a subtree deactivation control mechanism which is open to evolutionary learning. It is hypothesised that this representation confers an adaptive advantage in dynamic environments over the standard subtree representation adopted in Genetic Programming. Results presented on benchmark dynamic problem instances provides evidence to support that such an adaptive advantage exists.**

I. Introduction

THERE have been a number of significant studies in Evolutionary Computation applied to Dynamic problem environments (e.g., [1], [2], [3]). Research on the behaviour of Genetic Programming in dynamic environments, however, has been largely overlooked to date relative to the amount of work being undertaken in static environments [4]. Despite the existence of a recent Special Issue in the Genetic Programming Journal on Dynamic environments [5], none of the four articles actually dealt with GP directly (e.g., [6]). While some applications in dynamic environments have been undertaken in recent years (e.g., [7], [8], [9], [10], [11], [12], [13]), there has been little analysis of the behaviour of GP in these environments with the two main examples having examined bloat [14] and constant generation [4].

As the natural process of evolution operates in a non-stationary environment it seems natural that evolutionary algorithms such as Genetic Programming could present a useful tool to solve problems in these challenging environments.

In this paper we examine representational adaptation in a grammatical form of Genetic Programming, Grammatical Evolution. We do this by incorporating an extension to the standard Genetic Programming subtree representation that allows individuals to deactivate and (re)activate each subtree in the solution. When a subtree is deactivated it returns a functionally neutral value to its parent node. This is operationalised through the use of the identity function for each element of the function set. For example, when a subtree that is rooted in a parent node that is a multiplication operator is deactivated, it will return `1.0`. The net effect being whatever is returned from the second subtree of that parent node will be the overall value returned by the parent subtree.

A deactivated subtree is just that, *deactivated*, such that it can easily be later reactivated at a future time. The deactivated subtree is not removed from the individual, rather it is a kind of dormant memory which can later be reactivated. Such a feature may prove useful in the case of a dynamic environment where learnt knowledge that is not useful at this time may be reused at a later point in time when the environment changes. Through the use of a grammatical form of Genetic Programming this representational extension can be easily achieved through a simple extension to the input grammar.

Following a brief introduction to Grammatical Evolution in Section II, we present the subtree (de)activation grammars in Section III. Section IV details the results before finishing the paper with Conclusions and Future Work in Section V.

Michael O'Neill, Anthony Brabazon and Erik Hemberg are with the Natural Computing Research & Applications Group, University College Dublin, Ireland (email: m.oneill@ucd.ie, anthony.brabazon@ucd.ie, erik.hemberg@ucd.ie).

II. Grammatical Evolution

Grammatical Evolution (GE) [15] is a grammar-based form of Genetic Programming [16], [17], [18], [19], [20] that can be used to evolve computer programs, rule-sets, or more generally sentences in any language. Rather than representing the programs as syntax trees (as in GP), a linear genome representation is used in conjunction with a grammar. Each individual (genome), a variable-length binary string, contains in its codons (groups of bits) the information to select production rules from a Backus Naur Form (BNF) grammar. BNF is a notation that represents the language in the form of production rules. It is comprised of a set of non-terminals that can be mapped to elements of a set of terminals, according to the production rules. When tackling a problem with GE, a suitable BNF (Backus Naur Form) grammar definition must initially be defined. The BNF can be either the specification of an entire language or, perhaps more usefully, a subset of a language geared towards the problem at hand.

In GE, a BNF definition is used to describe the output language to be produced by the system. BNF is a notation for expressing the grammar of a language in the form of production rules. BNF grammars consist of *terminals*, which are items that can appear in the language, e.g. binary boolean operators `and`, `or`, `xor`, and `nand`, unary boolean operators `not`, constants, `true` and `false` etc. and *non-terminals*, which can be expanded into one or more terminals and non-terminals.

As the BNF definition is a plug-in component of the system, it means that GE can produce code in any language thereby giving the system a unique flexibility.

There have been a number of extensions to GE in recent years in terms of the grammars adopted, the search engine employed and even variants on the mapping process itself

(e.g., see the following sources [25], [22], [26], [27], [4], [23], [21]).

III. Subtree Activation Control Grammars

The problem domain examined is that of symbolic regression in a dynamic environment where the target function is undergoing change. The standard and deactivation grammars employed in this study are presented below in Figures 1 and 2. The grammars are deliberately minimalist in order to focus our analysis on any effect produced by the deactivation mechanism alone.

The evolved functions are evaluated as scheme s-expressions and are hence presented in their lambda calculus form. The deactivation grammar illustrates how the subtree deactivation mechanism is implemented. The deactivated subtree is functionally neutral with respect to the root it is nested in by returning the identity of the root operator (1.0 in the case of multiplication and 0.0 in the case of addition). A standard context free grammar can be adapted to represent this relationship by specifying specific expressions that are allowed for each root type. This convenience of the grammatical representation is the primary motivation for adopting a Grammatical approach to Genetic Programming such as GE in this study. Indeed, subtree deactivation could be incorporated easily into most grammar-based forms of Genetic Programming (e.g., [28], [29], [30], [31], [32], [33]) as well as standard Genetic Programming. If we examine the derivation trees produced by these grammars the net effect of the deactivation condition results in the ability of each subtree in a solution to be explicitly deactivated or (re)activated by modifying the state of the <deactivate> non-terminal, which is under genetic control in this study.

```
<code> ::= (define guess (lambda (x) (* <expr> <expr>)))
         | (define guess (lambda (x) (+ <expr> <expr>)))

<expr> ::= (* <expr> <expr>)
         | (+ <expr> <expr>)
         | <var>

<var> ::= x
```

Fig. 1. Standard control grammar producing prefix expressions.

```
<code> ::= (define guess (lambda (x) (* <*expr> <*expr>)))
         | (define guess (lambda (x) (+ <+expr> <+expr>)))

<expr> ::= (* <*expr> <*expr>)
         | (+ <+expr> <+expr>)
         | <var>

<*expr> ::= (if <deactivate> <expr> 1)

<+expr> ::= (if <deactivate> <expr> 0)

<var> ::= x

<deactivate> ::= #t | #f
```

Fig. 2. Subtree Activation/Deactivation grammar producing prefix expressions.

Figs. 3 and 4 gives a visual representation of the effect of subtree deactivation. Instead of a subtree being lost from the population it is temporarily deactivated, and can easily be reintroduced into the individual by modifying the state of the deactivation switch as implemented by the <deactivate> non-terminal. Effectively each subtree represents two possible states that can be evaluated.

IV. Experimental Setup & Results

There are two hypotheses under examination in this study. The first hypothesis tested is stated as follows:

H_0: There is no difference in performance between subtree deactivation and a standard subtree representation.

H_1: Subtree deactivation confers a performance advantage in a dynamic environment.

Two forms of the subtree deactivation setup are examined. The first simply adopts the deactivation grammar as specified in Figure 2. The second setup adopts a directed mutation operator, which targets the deactivation switch. Following from this a second hypothesis is tested as follows:

H_0: There is no difference in performance between subtree deactivation with and without directed mutation towards the deactivation sites.

H_2: Directed mutation towards subtree deactivation sites confers an adaptive advantage in terms of a more robust performance measured by improved fitness in a dynamic environment.

Oscillating targets were employed, changing every 20 generations for a total of 100 generations with a population size of 500. Probability of crossover (variable-length one point on the integer chromosome) is 0.9, probability of codon integer mutation is 0.01 (32 bit integer encoding). Tournament selection with tournament size of 3, and a rank replacement strategy where the children and parents are pooled with the top ranking population size candidate solutions been propagated to the subsequent generation. In the case of the directed mutation setup, where mutations are directed at the deactivation control switch, probabilities of 0.5, 0.25 and 0.125 were tested. The standard mutation operator was also allowed in this setup at the rate of 0.01. In the case of the directed mutation towards the deactivation sites the integer codon value used to expand each <deactivate> non-terminal is the target for mutation. If a <deactivate> codon is to be mutated the value 1 is added to its current state, and as we are always choosing expansion from two possible states the switch state will always be inverted when mutation occurs. It is trivial to keep track of the codon responsible for mapping each <deactivate> non-terminal by maintaining the derivation tree during the expansion of an individual during the genotype-phenotype mapping.

Three problem instances of dynamic symbolic regression were examined as detailed below.

a) Dynamic Symbolic Regression Instance #1:

$$f = x^2 + x^3 + x^5 \tag{1}$$

$$f = x^2 + x^5 \tag{2}$$

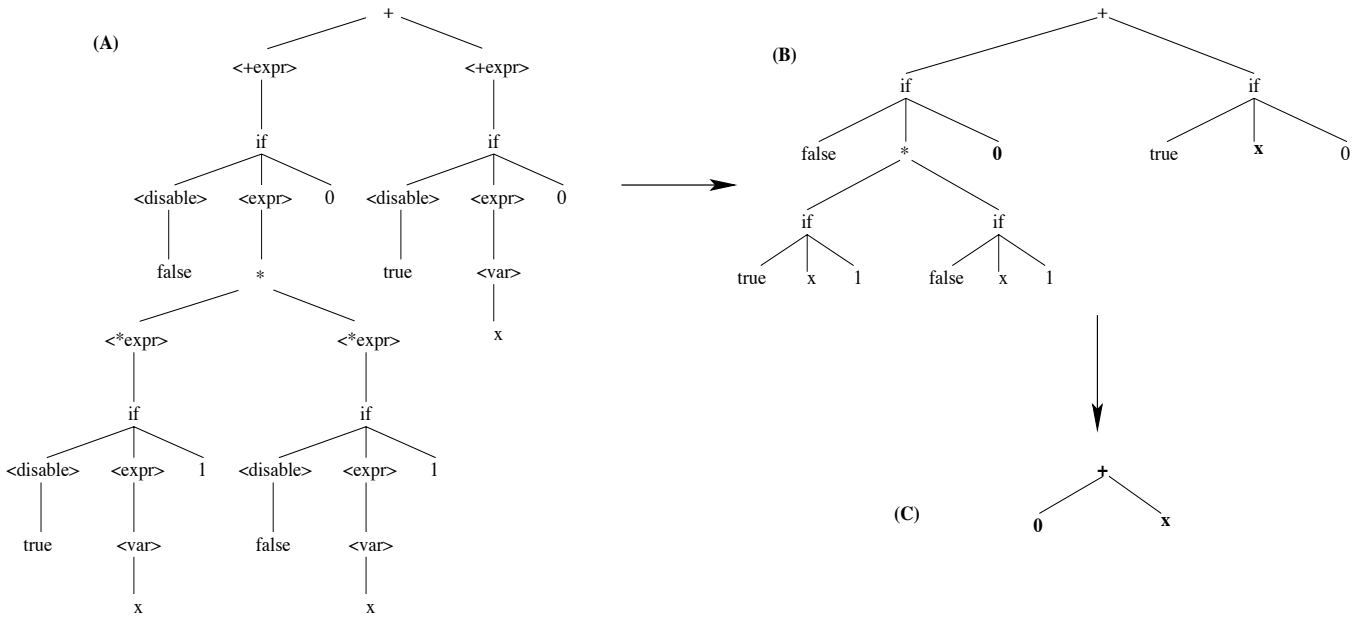


Fig. 3. An illustration of the subtree deactivation representation in operation on the derivation tree. Parts B and C illustrate the simplification of this derivation tree to it's corresponding parse tree

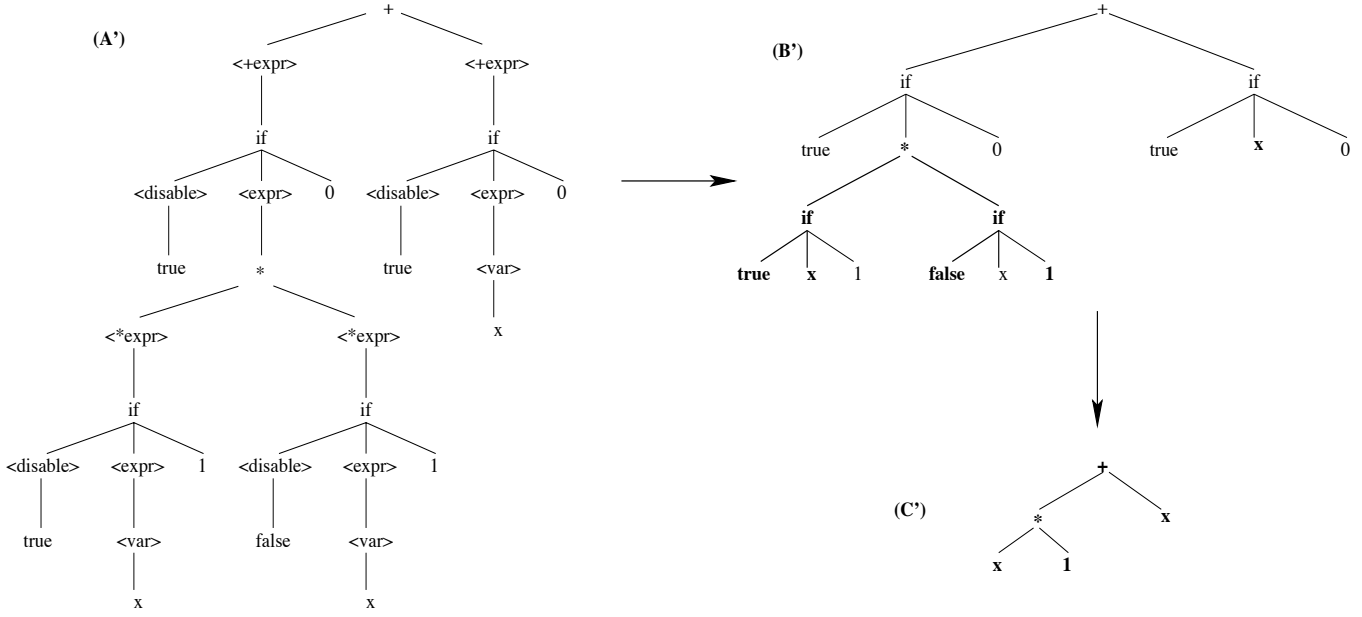


Fig. 4. The same derivation tree as in Fig. 3 but with the first deactivation switch turned off (A'). Parts B' and C' illustrate the simplification of this derivation tree to it's corresponding parse tree.

b) Dynamic Symbolic Regression Instance #2: The two targets are:

$$f = x + x^2 + x^3 + x^4 + x^5 \tag{3}$$

$$f = x^2 + x^3 + x^5 \tag{4}$$

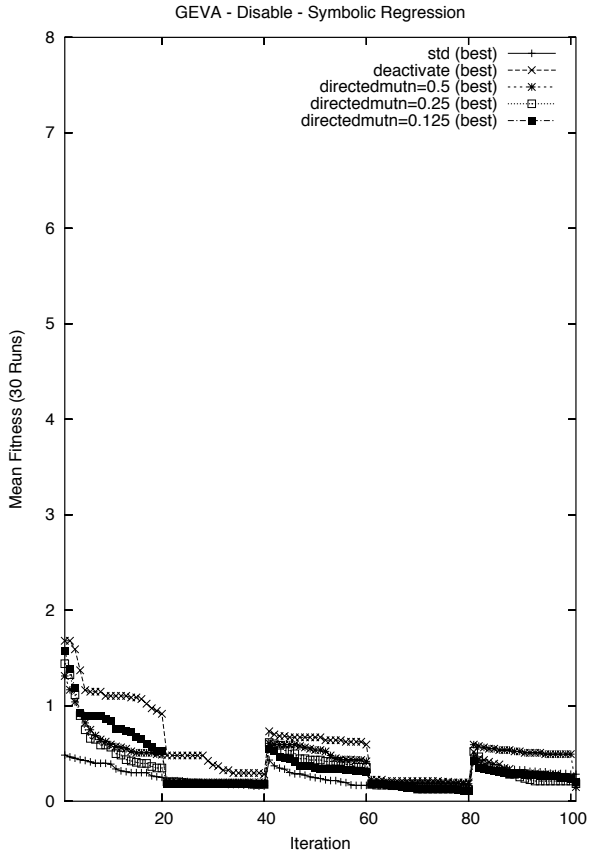
c) Dynamic Symbolic Regression Instance #3: At the start of each run two targets are randomly selected from:

$$f = x + x^2 + x^3 + x^4 \tag{5}$$

$$f = x + x^2 + x^3 \tag{6}$$

$$f = x + x^2 + x^4 \tag{7}$$

$$f = x^2 + x^3 + x^5 \tag{8}$$



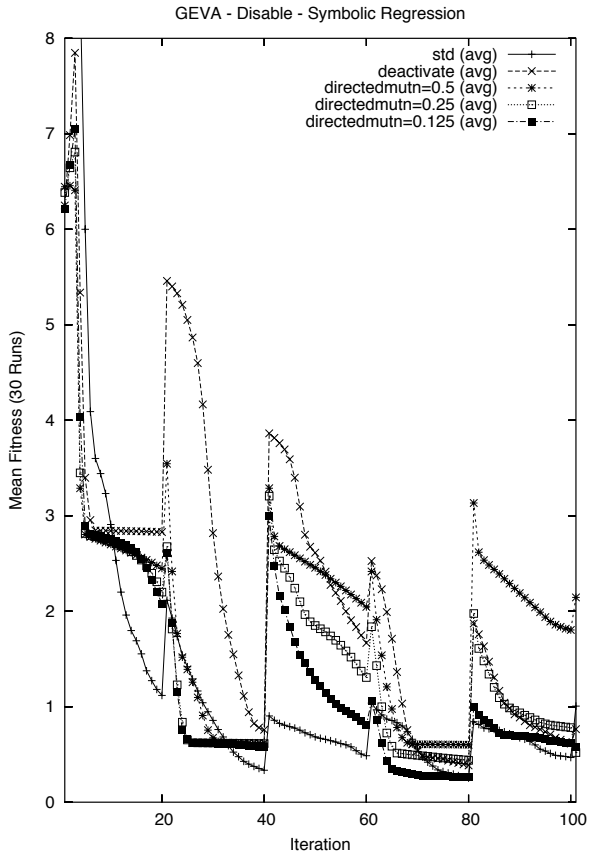
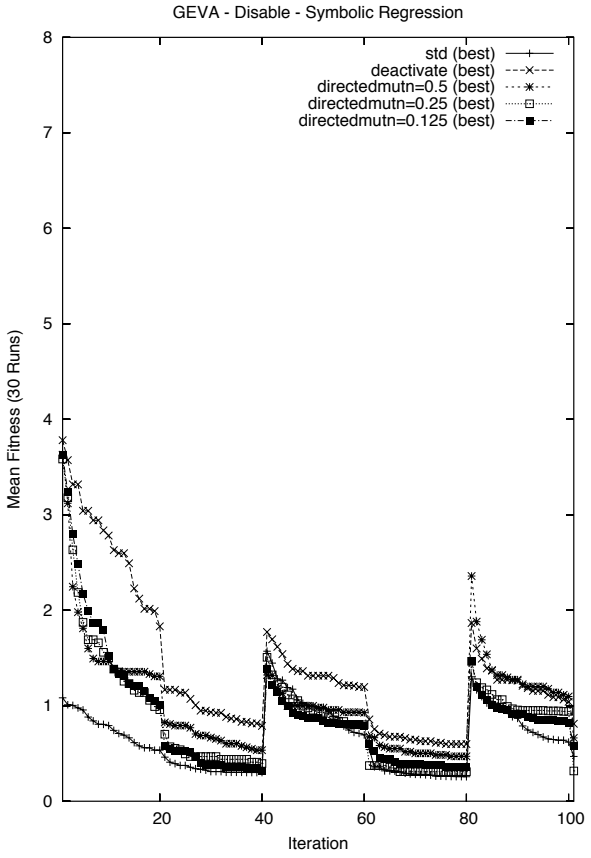


Fig. 5. The best and average fitness plots for problem instance 1 with period_length 20.



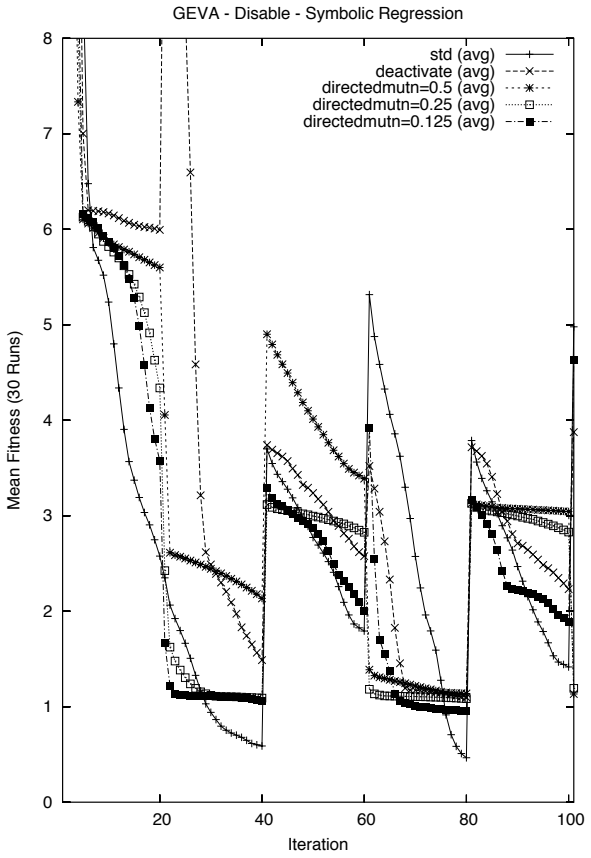


Fig. 6. The best and average fitness plots for problem instance 2 with period_length 20.

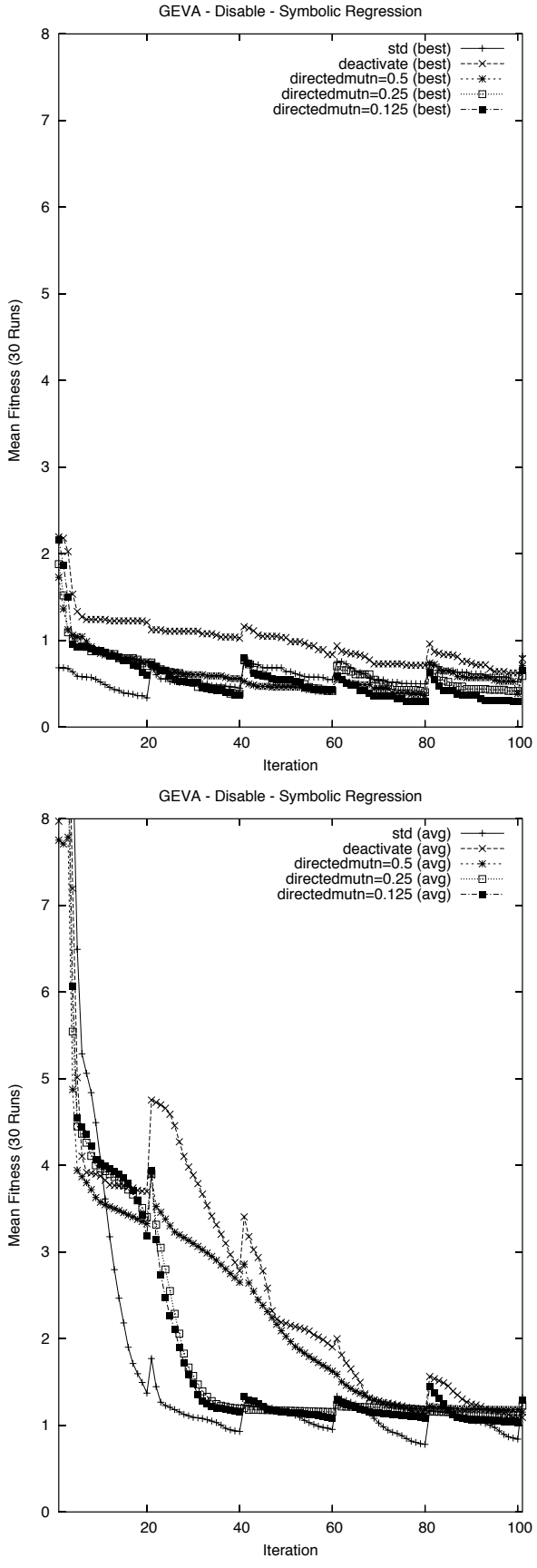


Fig. 7. The best and average fitness plots for problem instance 3 with period_length 20.

Fitness plots are presented in Figures 5, 6 and 7. A simplified *area under the curve* calculation for each fitness period and total area are provided for each of these figures in Tables I, II and III. This measure sums the error over each fitness period.

The results reflect an initial *warm up* period over the first few cycles of the fitness target during which time the performance of the standard representation is generally superior. However, following this *warm up* period the more adaptive deactivation representation demonstrates equal and in some cases a performance advantage over the static representation both in terms of the total error during each period of change and in terms of the fitness hit taken directly following a target change on Problem instances 1 and 3.

With respect to the first hypothesis (H_1) there is evidence to support rejecting the null hypothesis that there is no difference between the subtree deactivation and standard subtree representations on the two problem instances examined here.

The second hypothesis (H_2) asks if there is a performance difference between subtree deactivation with and without a directed mutation operator towards the deactivation switch. The evidence presented here provides strong evidence to support rejecting the null hypothesis that there is no difference between the presence and absence of directed mutation. There is a clear gain in performance in the presence of directed mutation.

A. Discussion

The observed performance of the deactivation representation is particularly impressive given the increase in search space size arising from the choice of the deactivation nonterminal for each subtree in an individual. It is also worth noting that no special initialisation strategy was adopted for either approach. Initialistion in each case was a pseudo random process. The consequence of this random initialisation is that on average 50% of the deactivation switches will be set to off. This means that in the initial population at least half of the subtrees of each solution that employ the deactivation grammar will not contribute to the fitness of an individual, and the effective sampling of the fitness space will be significantly less than in the case of the standard grammar.

In dynamic environments such as these pure fitness optimisation is no longer the solitary driving force with evolutionary robustness being of at least equal importance. Given that a real-world environment in which an adaptive algorithm could exist would perhaps ideally be a continously operating *live* evolutionary algorithm without restarts, the performance observed for the deactivation representation is particularly encouraging with these results suggesting a more consistent and robust performance might be possible.

V. Conclusions & Future Work

We introduced a subtree deactivation extension to the standard representation adopted in Genetic Programming. It was hypothesised that such an extension might confer an adaptive advantage in dynamic environments. Results presented here

TABLE I

AREA UNDER CURVE DATA (AVERAGED OVER 30 RUNS) DETAILED FOR EACH PERIOD OVER THE RUN WHERE THE TARGET CHANGES EVERY 20 GENERATIONS ON PROBLEM 1.

Problem 1

	Period					
Setup	**1**	**2**	**3**	**4**	**5**	**Total**
Std (best)	*7.19*	3.71	*5.10*	3.21	6.97	**26.17**
Deactivate (best)	23.54	7.71	13.12	4.20	6.81	**55.38**
DirectedMutn 0.5 (best)	13.78	3.96	10.37	3.79	10.62	**42.51**
DirectedMutn 0.25 (best)	12.67	3.74	9.05	2.86	*5.86*	**34.18**
DirectedMutn 0.125 (best)	16.82	*3.58*	7.56	*2.84*	6.05	**36.84**
Std (avg)	77.39	18.96	*13.76*	11.56	*13.96*	**135.64**
Deactivate (avg)	72.51	59.81	53.98	19.51	21.03	**226.84**
DirectedMutn 0.5 (avg)	*64.80*	21.56	48.98	17.39	46.21	**198.93**
DirectedMutn 0.25 (avg)	65.08	16.47	39.13	12.70	21.42	**154.80**
DirectedMutn 0.125 (avg)	65.81	*16.13*	28.69	*7.55*	15.03	**133.20**

TABLE II

AREA UNDER CURVE DATA (AVERAGED OVER 30 RUNS) DETAILED FOR EACH PERIOD OVER THE RUN WHERE THE TARGET CHANGES EVERY 20 GENERATIONS ON PROBLEM 2.

Problem 2

	Period					
Setup	**1**	**2**	**3**	**4**	**5**	**Total**
Std (best)	*15.35*	*6.72*	20.35	*6.04*	*17.62*	**66.07**
Deactivate (best)	54.10	19.22	27.31	13.09	26.14	**139.85**
DirectedMutn 0.5 (best)	33.75	13.39	20.42	10.52	27.68	**105.76**
DirectedMutn 0.25 (best)	33.22	9.51	19.88	6.53	21.02	**90.16**
DirectedMutn 0.125 (best)	35.31	8.39	*18.43*	8.12	19.72	**89.97**
Std (avg)	*118.31*	*22.74*	*54.26*	51.54	53.61	**300.46**
Deactivate (avg)	143.89	87.81	63.68	33.09	61.25	**389.72**
DirectedMutn 0.5 (avg)	133.19	49.71	80.59	24.44	62.65	**350.58**
DirectedMutn 0.25 (avg)	129.33	25.07	59.64	*22.16*	60.84	**297.03**
DirectedMutn 0.125 (avg)	127.78	*22.74*	54.38	26.10	*52.14*	**283.13**

TABLE III

AREA UNDER CURVE DATA (AVERAGED OVER 30 RUNS) DETAILED FOR EACH PERIOD OVER THE RUN WHERE THE TARGET CHANGES EVERY 20 GENERATIONS ON PROBLEM 3.

Problem 3

	Period					
Setup	**1**	**2**	**3**	**4**	**5**	**Total**
Std (best)	*10.09*	10.28	12.83	11.58	13.18	**57.95**
Deactivate (best)	27.75	21.68	19.99	15.45	15.65	**100.53**
DirectedMutn 0.5 (best)	18.94	12.26	*9.12*	8.95	12.38	**61.64**
DirectedMutn 0.25 (best)	18.92	10.62	10.63	10.56	10.12	**60.85**
DirectedMutn 0.125 (best)	19.41	*10.17*	10.81	*7.74*	*8.21*	**56.35**
Std (avg)	90.91	*22.66*	*22.36*	*20.72*	*22.12*	**178.77**
Deactivate (avg)	95.87	76.79	47.32	27.45	26.77	**274.20**
DirectedMutn 0.5 (avg)	*85.05*	62.07	41.54	26.11	24.98	**239.75**
DirectedMutn 0.25 (avg)	92.88	37.78	23.38	24.05	24.07	**202.16**
DirectedMutn 0.125 (avg)	94.67	35.52	23.42	23.13	23.76	**200.49**

suggest that such an adaptive advantage can exist in certain types of dynamic environments, and consequently warrants further investigation. Future work will include an analysis of subtree deactivation on a number of different problem environments to test the generalisation of the results beyond the symbolic regression instances examined here.

The developmental nature of the grammatical form of Genetic Programming adopted in this study, Grammatical Evolution, provides us with another potentially powerful manner to approach adaptation in dynamic environments. In particular, it will be possible to allow a form of epigenetic and developmental learning to occur as a solution is being generated from its embryonic start symbol. It is trivial to implement subtree (de)activation that can operate on much shorter time scales than the evolutionary one present in this case. For example, by allowing feedback from the environment (such as fitness of the parent) this could be allowed to alter the switches that are present. If a change in the conditions of the parent are detected this information can be passed on epigenetically through the switches, thus (de)activating subtrees that were previously (de)activated. We also wish to examine a form of local search where we sample an individual `N` times by toggling random switches.

ACKNOWLEDGEMENT

This research is based upon works supported by Science Foundation Ireland under Grant No. 06/RFP/CMS042.

REFERENCES

[1] Branke, J. (2001). *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers.

[2] Morrison, R.W. (2004). *Designing Evolutionary Algorithms for Dynamic Environments*. Springer.

[3] Branke, J. (2005). Towards and analysis of Dynamic Environments. In *Proc. of the Genetic and Evolutionary Computation Conference GECCO 2005*, pp.1433-1440. Vol.2. ACM Press.

[4] Dempsey I. (2007). Grammatical Evolution in Dynamic Environments, *PhD Thesis*, University College Dublin

[5] Yang, S., Ong, Y-S, Jin, Y. (2006). Special Issue on Evolutionary Computation in Dynamic and Uncertain Environments. Genetic Programming and Evolvable Machines, Vol.7 No.4.

[6] Wang, Y., Wineberg, M. (2006). Estimation of evolvability genetic algorithm and dynamic environments. Genetic Programming and Evolvable Machines, Vol.7 No.4, pp.355-382.

[7] Wagner, N., Michalewicz, Z., Khouja, M., McGregor, R.R. (2007). Time Series Forecasting for Dynamic Environments: The DyFor Genetic Program Model. IEEE Transactions on Evolutionary Computation, Vol.11 No.4, pp.433-452.

[8] Tsang, E., Yung, P., Li, J. (2004). EDDIE-Automation, a decision support tool of financial forecasting. Decision Support Systems, Vol.37 No.4., pp.559-565.

[9] Yan, W., Clack, C.D. (2006). Behavioural GP diversity for dynamic environments: an application in hedge fund investment. In Proceedings of GECCO 2006 Genetic and Evolutionary Computation Conference, Seattle Washington, pp.1817-1824. ACM Press.

[10] Hansen, J.V., Lowry, P.B., Meservy, R.D., McDonald, D.M. (2006). Genetic Programming for Prevention of Cyberterrorism through dynamic and evolving intrusion detection. Decision Support Systems, Vol.43 No.4, pp.1362-1374.

[11] Jakobović, D., Budin, L. (2006). Dynamic Scheduling with Genetic Programming. LNCS 3905 Proceedings of EuroGP 2006 the European Conference on Genetic Programming, pp.73-84. Springer.

[12] Kibria, R.H., Li, Y. (2006). Optimising the Initialisation of Dynamic Decision Heuristics in DPLL SAT Solvers using Genetic Programming. LNCS 3905 Proceedings of EuroGP 2006 the European Conference on Genetic Programming, pp.331-340. Springer.

[13] Hu, J., Goodman, E. (2004). Topological Synthesis of Robust Dynamic Systems for Sustainable Genetic Programming. Genetic Programming Theory and Practice. Chapter 9. Springer.

[14] Langdon, W.B., Poli, R. (1998). Genetic Programming Bloat with Dynamic Fitness. LNCS 1391 Proceedings of EuroGP 1998 the First European Workshop on Genetic Programming, pp.96-112. Springer.

[15] O'Neill, M., Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers.

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

[17] Koza, J.R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press.

[18] Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D. (1998). *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann.

[19] Koza, J.R., Andre, D., Bennett III, F.H., Keane, M. (1999). *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufmann.

[20] Koza, J.R., Keane, M., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

[21] O'Neill M., Ryan C. (2004). Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code, *EuroGP* 2004 138-149 LNCS 3003 Springer Coimbra, Portugal.

[22] Brabazon, Anthony and O'Neill, Michael (2006), *Biologically Inspired Algorithms for financial Modelling*, Springer.

[23] O'Neill M., Brabazon A. (2005). mGGA: The Meta-Grammar Genetic Algorithm. *LNCS 3447 EuroGP 2005*, pp.311-320. Springer.

[24] Hemberg E., Gilligan C., O'Neill M., Brabazon A. (2007). A Grammatical Genetic Programming Approach to Modularity *In EuroGP* 2007 Springer Ebner M., et al Valencia, Spain

[25] O'Neill, M., Brabazon, A. (2005). Recent Adventures in Grammatical Evolution. In Proceedings of CMS 2005 Computer Methods and Systems. Vol.1, pp.245-253, November 2005, Krakow, Poland.

[26] Dempsey I., O'Neill M., Brabazon A. (2007). Constant Creation with Grammatical Evolution. International Journal of Innovative Computing and Applications, pp.23-38 Vol.1 No.1

[27] O'Neill, M. and Brabazon, A. (2006). Grammatical Swarm: The Generation of Programs by Social Programming. Natural Computing, pp.443-462 Vol.5 No.4

[28] Whigham, P.A. (1996). Grammatical Bias for Evolutionary Learning. PhD Thesis. University of New South Wales, Australian Defence Force Academy.

[29] Wong, M.L., Leung, K.S. (2000). Data Mining using Grammar Based Genetic Programming and Applications. Kluwer Academic Publishers.

[30] Gruau, F. (1994). Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. PhD Thesis. Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon. France.

[31] Paterson, N. (2002). Genetic Programming with Context-Sensitive Grammars. PhD Thesis. Saint Andrew's University.

[32] Hoai, X.H. (2004). A Flexible Representation for Genetic Programming from Natural Language Processing. PhD Thesis. Australian Defence Force Academy, University of New South Wales.

[33] Shan, Y. (2005). Program Distribution Estimation with Grammar Models. PhD Thesis. University of New South Wales.