An Investigation on the Roles of Insertion and Deletion Operators in Tree Adjoining Grammar Guided Genetic Programming

Nguyen Xuan Hoai and RI Mc Kay
Dept. of Information Technology and Electrical Engineering,
Australian Defence Force Academy, Univeisty of New South Wales
ACT 2600 Australia.
E-mail: x.nguyen@adfa.edu.au; rim,daryl,abbass@cs.adfa.edu.au

Abstract- In this paper, we investigate the roles of insertion and deltion as mutation operators and local search operators in Tree Adjoining Grammar Guided Genetic Programming (TAG3P) system [13]. The results show that, on three standard problems, these operators are better than the sub-tree mutation originally used in [13, 14]. Moreover, insetion and deltion can act as local search operators and help TAG3P to solve problems with only small population sizes.

I. INTRODUCTION

Tree adjoining grammar guided genetic programming (TAG3P) [13, 14] (some of its only forms were presented in [11, 12]) is a genetic programming system that uses tree adjoining grammars (TAG) as the formalisms to dictate its language bias. We argued in [13] that one of the advantages of using TAG-based representation is the 'feasibility' (described in section 2) in TAG derivation trees, which allows us to design many types of operators [13]. Among those operators, insertion and deletion arise naturally as structural mutation operstors. In a recent work [15], we have shown how TAG-based representation coupled with insetion and deletion can soften significantly the inherent structural search difficulty in genetic programming [5, 6].

In this paper, we empirically investigate the use of insertion and deltion operators in the context of tree adjoining grammar guided genetic programming (TAG3P) and compared to the results to TAG3P using subtree mutation opertors as in [13, 14].

The paper is, therefore, organized as follows. In section 2 we brief reinstroduce the concepts of tree adjoining grammars and TAG3P as well as insertion and deltion operators. The experiments to investigate the roles of insertion and deletion in TAG3P will be presented and discussed in section 3. Section 4 concludes the paper and highlight some future work.

II. TAG-BASED REPRESENTATION FOR GP

In this section, we first give the definitions of tree adjoining grammars (TAGs) and their derivation trees. Then, we describe how TAG-derivation trees can be used for genetic programming as in [13].

II. 1 Tree Adjoining Grammars

Joshi and his colleagues in [8] proposed tree-adjunct grammars, the original form of tree adjoining grammars (TAG). Adjunction was the only tree-rewriting operation. Later, the substitution operation was added and the new formalism became known as TAG. Although the addition of substitution did not change the strong and weak generative power of tree adjunct grammars (their tree and string sets), it compacted the formalism with fewer elementary trees [9].

TAGs are tree-rewriting systems, defined in [9] as a 5-tuple (T, V, I, A, S), where T is a finite set of terminal symbols; V is a finite set of non-terminal symbols (T ⊠ V = ⊠); S ⊠ V is a distinguished symbol called the start symbol; and E = I ⊠ A is a set of elementary trees (initial and auxiliary respectively). In an elementary tree, interior nodes are labeled by non-terminal symbols, while nodes on the frontier are labeled either by terminal or non-terminal symbols. The frontier of an auxiliary tree must contain a distinguished node, the foot node, labeled by the same non-terminal as the root. The convention in [9] of marking the foot node with an asterisk (*) is followed here. With the exception of the foot node, all non-terminal symbols on the frontier of an elementary tree are terminal or marked as ⊠ for substitution. Initial and auxiliary trees are denoted ⋄ and ≋ respectively. A tree whose root is labeled by X is called an X-type tree. Figure 1 shows some examples of initial and auxiliary trees.

The key operations used with tree-adjoining grammars are the adjunction and substitution of trees. Adjunction builds a new (derived) tree ∵ from an auxiliary tree ≋ and a tree ⋄ (initial, auxiliary or derived). If tree ⋄ has an interior node labeled A, and ≋ is an A-type tree, the adjunction of ≋ into ⋄ to produce ∵ is as follows: Firstly, the sub-tree $⋄_1$ rooted at A is temporarily disconnected from ⋄. Next, ≋ is attached to ⋄ to replace the sub-tree. Finally, $⋄_1$ is attached back to the foot node of ≋. ∵ is the final derived tree achieved from this process. Adjunction is illustrated in Figure 2.

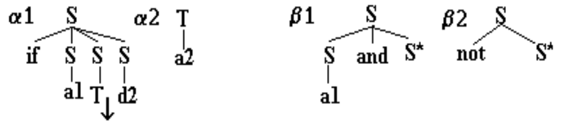

Figure 1. Some examples of initial and auxiliary trees.

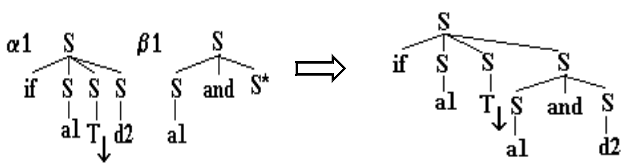

Fig.ure2. Adjunction.

In substitution, a non-terminal node on the frontier of an elementary tree is substituted with another initial tree with a root labelled with the same non-terminal symbol. Substitution is illustrated in Figure 3.

The tree set of a TAG can be defined as follows [9]:
T_G = {all tree t: t is completed and t is derived from some initial S-trees through adjunctions and substitutions}.

Where a tree t is completed, if t is an initial tree and all of the leaf nodes of t are labelled by terminal symbols. The language generated by the TAG G is defined as

L_G = {w ∈ T*: w is the yield of some tree t ∈ T_G}.

In TAG, there is a distinction between derivation and derived trees. A derivation tree in TAG [9, 17, 20, 21] is a tree-structure, which encodes the history of derivation (substitutions and adjunctions) to produce the derived tree. Each node is labelled by an elementary tree name: the root must be labelled by an ⋄ tree name, and the other nodes with either an ⋄ or β tree. The links between a node and its offspring are marked by addresses for adjunctions and substitutions. Figure 4 illustrates the derivation and derived trees in TAGs (the discontinuous lines mean substitutions).

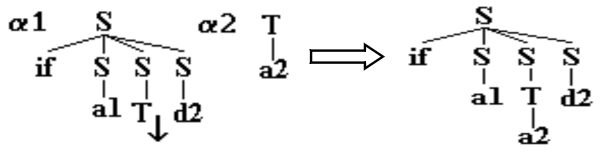

Figure 3. Substitution.

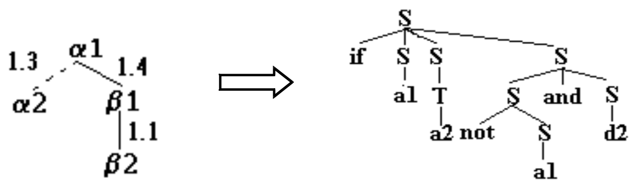

Figure 4. Examples of a derivation tree and derived tree in TAGs.

The set of languages generated by TAGs (called TAL) is a superset of the context-free languages generated by CFGs; and is properly included in indexed languages [9]. More properties of TAL can be found in [9].

One special class of TAGs is lexicalized TAGs (LTAGs) [9], in which each elementary tree of an LTAG must have at least one terminal node. It has been proven that there is an algorithm which, for any context-free grammar G, generates a corresponding LTAG G_{lex} that generates the same language and tree set as G (G_{lex} is then said to strongly lexicalize G) [9]. The derivation trees in G are the derived trees of G_{lex}.

II.2 Tree Adjoining Grammar Guided Genetic Programming (TAG3P)

The algorithm in [9, 18, 19] to find an LTAG to strongly lexicalize a CFG is based on the ideas of separation between the recursive part (structure) and non-recursive part (lexicon) of the CFG. In [9, 18, 19], the only operation necessary in the resultant LTAG is adjunction. However, substitution can be added to make the elementary set more compact [9]. Moreover, it is possible to encode the non-recursive parts of the grammar purely as substitution trees. In so doing, the initial tree used for substitution cannot be adjoined by other auxiliary trees: a process that simplifies the structure of derivation trees in LTAGs while maintaining their generative powers. Consequently, on the structure of LTAG derivation trees, substitution becomes an in-node operation and can be ignored to simplify the discussion of this paper (in fact, one can choose to entirely ignore substitution in implementing a TAG-based representation, at the cost of increasing the number of elementary trees). Figure 5 depicts this type of LTAG derivation tree (supposing each elementary tree has two adjoining addresses – i.e. the maximum arity is 2).

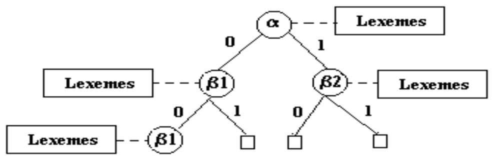

Figure 5. Derivation tree structure for TAG-based representation. The squares means there is no tree adjoining to that address (a NULL node).

In TAG3P [13, 14], the derivation tree in LTAG was used as genotype structure. TAG3P uses a genotype-to -phenotype map and can handle problems with context-sensitive syntactical constraints, context-free syntactical constraints, or (as in standard GP) no syntactical constraints. In the first case, an LTAG grammar G_{lex} is used on its own as the formalism for language bias declaration. The phenotype is the derived tree of G_{lex}. In the second case, the context-free grammar (CFG) G is used to generate the strongly lexicalised LTAG G_{lex}. The derivation tress of G_{lex} is used as the genotype, and the phenotype in that case is the derivation tree of G (derived tree of G_{lex}). In the final case, from a description of a GP set of functions and terminals, a context-free grammar, G, is created according to [22] (page 130). It was proven in [22] that there is a one to one map between the derivation trees of G and the expression trees in GP. The mapping schema can be summarized in figure 6 as follows where the second phase of the map is optional.

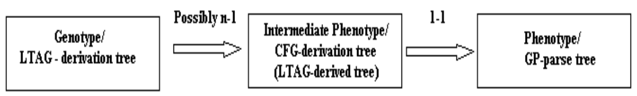

Figure 6. Schema for Genotype-to-Phenotype map in TAG-based Representation.

Other components of TAG3P are as follows [13]:

Parameters: minimum size of genomes (MIN_SIZE), maximum size of genomes (MAX_SIZE), size of population (POP_SIZE), maximum number of generations (MAX_GEN) and probabilities for genetic operators.

Initialization procedure: Each individual is generated by randomly growing a derivation tree in G_{lex} to a size randomly chosen between MIN_SIZE and MAX_SIZE (unlike most GP systems, which use depth bounds). Because of TAG feasibility, this always generates valid individuals of exact size. An alternative ramped half-and-half initialization generates half of the derivation trees randomly but of full shape.

Fitness Evaluation: an individual derivation is first mapped to the derived CFG tree. The expression defined by the derived tree is then semantically evaluated as in grammar guided genetic programming (GGGP) [22], or translated further into the parse tree and then be evaluated as in GP [4, 10].

Main Genetic operators: sub-tree crossover and sub-tree mutation.

In sub-tree crossover, two individuals are selected based on their fitness. Points with the same adjunction label are randomly selected within each tree and the two sub-trees are exchanged. If no such points are found, the two individuals are discarded and the process is repeated until a bound is exceeded.

In sub-tree mutation, a point in the derivation tree is chosen at random and the sub-tree rooted at that point is replaced by a newly generated sub-derivation tree.

II.3 Insertion and Deletion Operators

The derivation tree structure in LTAG has an important property: when growing it, one can stop at any time, and the derivation tree and the corresponding derived tree are still valid. In other words, the derivation tree in LTAG is a non-fix-arity tree structure (Catalan tree). The maximal arity (number of children) of a node is the number of adjoining addresses that are present in the elementary tree of that node. If this arity is n, the node can have 0, 1,..., or n children.

In [13], this property was called feasibility. Feasibility allows us to design and implement many other new search operators in TAG3P which would not be possible in standard GP and other GGGP systems [4, 7, 10, 16, 22-24], including bio-inspired ones [13]. In particular, insertion and deletion operators arise naturally from this TAG-based representation. In insertion, a random NULL node in the LTAG-derivation tree is replace by a new node that can adjoin to the adjoining address of the corresponding parent node. Conversely, deletion randomly deletes a node that has all NULL children in the LTAG-derivation tree (i.e. a leaf node). Insertion and deletion simulate the growth and shrinkage of a natural tree. The change in genotype structure (and consequently in phenotype structure) is small. Figure 7 illustrates how insertion and deletion work.

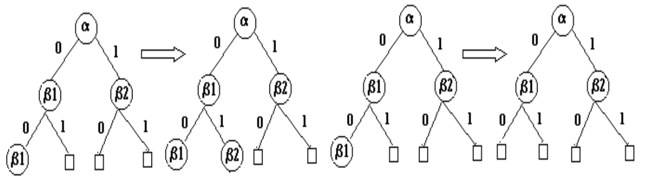

Fig.ure 7. Examples of insertion (on the left) and deletion (on the right).

Recently, we have shown that TAG-based representation coupled with insertion/deletion operators can help to soften significantly the structural search difficulty in GP [15]. Moreover, since compared to sub-tree mutation, insertion and deletion operators make relatively small change to the genotype in TAG3P (although the change in the phenotype fitness might be large), it is natural to raise the question of the use of these operators as mutation operators for TAG3P or local search operators in hybrid with genetic search in TAG3P system.

III. EXPERIMENTS AND RESULTS

In order to investigate the potential roles of insertion and deletion operators in TAG3P, we tried them on three problems, namely, 6-multiplexer, symbolic regression, and digital circuit synthesis.

In the 6-multiplexer problem [10], a 6-multiplexer uses two address lines to output one of four data lines and the task is to learn this function from its 64 possible fitness cases using function set F={IF, AND, OR, NOT} and terminal set {a0, a1, d0, d1, d2, d3}. In the symbolic regression problem [10], the task is to learn the quadtic function: $X^4+X^3+X^2+X$ from 20 sample points in [-1..1]; the function and terminal set are F={+,-,*,/,sin, cos, exp, rlog} and T={X}. They were chosen as frequently-used GP test-beds. In the last problem, the task is to synthesize digital combinatorial circuits from its input-output description using 1-line control multiplexers as basic building blocks [1-3]. As argued in [1], the use of only 1-line control multiplexer as the basic building block help manufacturers reduce the producing cost since they can produce only one basic gate (1-line multiplexer) in a (possible) massive scale. In [1-3], two types of 1-line control multiplexers were used, namely, class A and class B. However, in terms of generative power, one class is enough and therefore will be assumed in this paper (class A). It is also noted that, when only one class is used, the problem, in effect, is equivalent to the problem of inducing Boolean decision trees. The Boolean target function used in the experiment of this paper is the function in example 2 in [1-3], where the function has four input, one out put, empty dc-det, and on-set as {0,4,5,6,7,8,9,10,15}.

III.1 Experiment Design

To investigate the roles of insertion and deletion as mutation operators, for each problem, we use three sets of runs. On the first, we run TAG3P with sub-tree crossover and sub-tree mutation (base runs) (POP500CS); In the second, we run TAG3P with sub-tree crossover as the sole genetic operator (POP500CO). In the last, we run TAG3P with sub-tree crossover and insertion/deletion as mutation operators (POP500ID). The aim is to separate out the effect of using different mutation operators from the power of crossover alone. The POP_SIZE and MAX_GEN were 500 and 51 respectively in all sets of runs.

To investigate the roles of insertion and deletion as local search operators hybridizing with sub-tree crossover and sub-tree mutation in TAG3P, we designed two sets of runs. In the first, the POP_SIZE and MAX_GEN were 50 and 51 respectively, while the number of local search steps was 10. In effect, each run will have the same maximum number of fitness evaluations with one of POP500CS (or POP500CO, POP500ID). To find out whether the hybridization with genetic operators really works, we further divide this set of runs into two. Operators was turned on (POP50ON) in one set of runs and off in the other (POP50OFF). Similarly, on the second set of runs, the POP_SIZE and MAX_GEN were 10 and 51, while the number of local search steps was 50. We also designed POP10ON and POP10OFF set of runs with similar meaning with the first set of runs. The purpose of using two different population sizes is to investigate the trade-off between decreasing population size and increasing the number of local search steps. In all runs, the local search strategy was stochastic hill-climbing; and Lamarckian's inheritance was used (i.e. whenever a local search find a better individual in the neighborhood of an individual in the population, this individual is replaced by the new better individual).

For the sake of completion, for each problem, we also allocated a set of runs using purely stochastic hill-climbing (TAG-HILL) on TAG-based representation (i.e. no population and/or genetic operator were used). For each run with TAG-HILL, the maximum number of search steps was 25500 to make the maximum number of fitness evaluation s the same with other sets of runs above.

For each setting described above, 100 runs were allocated. For each problem, a random search (TAG-RAND) with sample size of 2550000 (i.e. the maximum number of fitness evaluations is equivalent to 100 runs in each setting) was also conducted. That makes the total of runs were 2700.

When they are used, the crossover and mutation rates were always set as 0.9 and 0.1; the MAX_SIZE were set as 40 for the 6-multiplexer and digital circuit design problems, and 20 for the symbolic regression problem.

The grammars G and G_{lex} for symbolic regression and 6-multiplexer were the same as in [13, 14], while G and G_{lex} for digital circuit design problem is as follows.

G={V={B, ATT}, T={c0,c1,c2,c3,0,1}, P,{B}}where the rule set P is defined as follows:

B → B ATT B.

B → 0 | 1

ATT → c0 |c1 | c2 |c3

G_{lex}={V={B, TL, ATT}, T={c0,c1,c2,c3,0,1}, I, A}, where I → A is depicted in Figure 8. TL and ATT are lexicons that can be substituted by one lexeme in {0,1} and in {c0,c1,c2,c3} respectively.

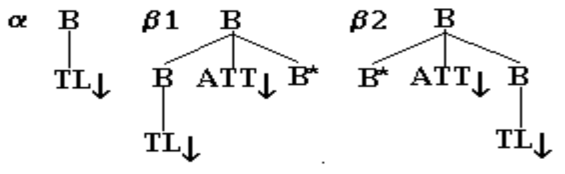

Figure 8. Elementary trees for G_{lex} in the digital circuit design problem.

III. 2 Results and Discussion

The results over 2700 runs for all three problems are presented in in tables 1, 2, and 3. The cumulative frequencies of the sets of runs that, at least, found a solution are depicted in Figure 9,10, 11 respectively.

The results show that, on all three problems, the combination of sub-tree crossover and insertion/deletion as mutation operators (POP500ID) turn out to be the best. For 6-multiplexer and digital circuit desgin problems, it outperformed statistically significantly (with ⋄=0.5%) POP500CS and POP500CO. For symbolic regresion, POP500ID was just slightly better (not statistical significance). That result can be explained by the power of sub-tree crossover operator on TAG-representation for that problem (with proportion of success of POP500CO was 95 %). Those results also suggest that, sub-tree mutation, which have been using in TAG3P since [13], is perhaps too disruptive.

On the investigation into the use of insertion/deletion as local search operators, the results show that POP50ON/OFF and POP10ON/OFF significantly better than POP500CS and POP500ID on the 6-multiplexer problem (with a statistical confidence level of 95%), while much worse on symbolic regression and digital circuit design problems. The results of TAG-HILL and POP500CO give an explanation for that. For 6-multiplexer problem, the landscape seems to be smoother for insertion/deletion operators than sub-tree crossover, while it is converse in the case of symbolic repgression and digital circuit design problems.

The results of TAG-HILL compared to POP500ID (especially on the 6-multiplexer problem) suggest that (even for problems that crossover alone performs badly) it is still essential to use it as well as the use of population (what we need is a suitable mutation operators).

The outperformance of POP50ON, POP10ON over POP50OFF, POP50ON indicate the usefulness of hybridization between genetic operators and local search using insertion/deletion. Furthermore, the supperior performance of POP50ON, POP10ON over POP500CS and POP500CO on the 6-multiplexer problem highlight that when the fitness landscape is smooth for insertion/deletion, they can be used efficiently as local search operators. The similar performance of POP50ON and POP10ON on that problem also indicate that we might be able to use small popultation size (and therefore save memory storage), to a certain extend, with an expense of increasing the length of local search to solve the problem reliably.

TABLE I. RESULTS ON THE 6-MULTIPLEXER.

Setting	Proportion of Successs (%)
POP500CS	38
POP500CO	32
POP500ID	89
POP50ON	78
POP50OFF	59
POP10ON	75
POP10OFF	72
TAG-HILL	61
TAG-RAND	1 in 2550000 samples

TABLE I I. RESULTS ON THE SYMBOLIC REGRESSION.

Setting	Proportion of Successs (%)
POP500CS	95
POP500CO	97
POP500ID	98
POP50ON	75
POP50OFF	16
POP10ON	74
POP10OFF	16
TAG-HILL	3
TAG-RAND	7 in 2550000 samples

TABLE I I. RESULTS ON THE DIGITAL CIRCUIT DESIGN.

Setting	Proportion of Successs (%)
POP500CS	70
POP500CO	84
POP500ID	91
POP50ON	34
POP50OFF	0
POP10ON	17
POP10OFF	0
TAG-HILL	0
TAG-RAND	0 in 2550000 samples

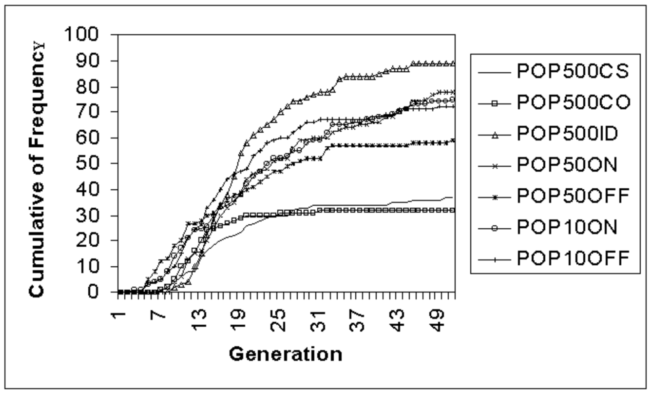

Fig.ure 9. Cumulative Frequencies on the 6-multiplexer problem.

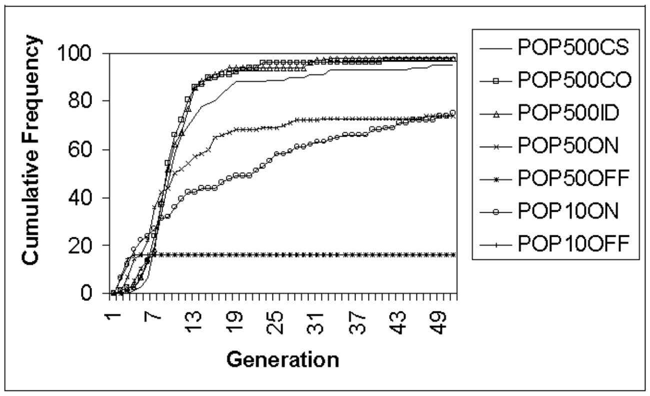

Fig.ure 10. Cumulative Frequencies on the symbolic regression problem.

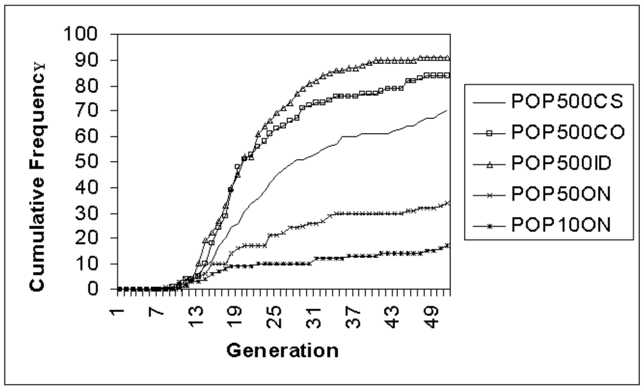

Fig.ure 11. Cumulative Frequencies on the digital circuit design problem.

IV. CONCLUSION

In this paper, we have reintroduced TAG-based representation. The insertion and deletion operators, which arise naturally from the representation was described in details. The investigation into the two possible roles of insertion/deletion in TAG3P was conducted and compared with TAG3P using sub-tree mutation operator.

The results suggest that, on the problems tried, insertion/deletion can be best used as mutation opreators regardless whether the fitness landscape is smooth for them or not. When the fitness landscape is smooth for insertion/deletion, they can potentially be used as local search operators in a hybridization with other genetic operators to archieve reliable performance even with very small population sizes.

In future, we are planning to invetigate the combination of insertion/deletion with other genetic oparater described in [13]. We are also developing a way to qualtify the smoothness of fitness landscape using insertion/deletion.

REFERENCES

[1] A.H. Aguirre, C. Coello-Coello, and B.P. Buckles, “A Genetic Programming Approach for Logic Function Synthesis by Mean of Multiplexers”, in *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*, IEEE Press, 1999, pp. 46-53.

[2] A.H. Aguirre, B.P. Buckles, and C. Coello-Coello, “Evolutionary Synthesis of Logic Function Using Multiplexers”, in *Proceedings of The 10th Conference on Smart Engineering Design (ANNIE 2000)*, ASME Press, 2000, pp. 311-316.

[3] A.H. Aguirre, B.P. Buckles, and C. Coello-Coello, “Gate-level Synthesis of Boolean Functions Using Binary Multiplexers and Genetic Programming”, at citeseer.nj.nec.com/309980.html, accessed 20/01/2003.

[4] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone, *Genetic Programming: An Introduction*, Morgan Kaufmann Pub, 1998.

[5] J.M. Daida, J.A. Polito 2, S.A. Stanhope, R.R. Bertram, J.C. Khoo, S.A. Chaudhary, and O. Chaudhri, “What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming”,. *Journal of Genetic Programming and Evolvable Hardware*, vol. 2, pp. 165-191, 2001.

[6] J.M. Daida, H. Li, R. Tang, and A.M. Hilss, “What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes”, in *Proceedings of GECCO 2003*, LNCS Springer-Verlag, 2003, pp.. 1665-1677.

[7] A. Geyer-Schulz, *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, Physica-Verlag, 1995.

[8] A.K. Joshi., L.S. Levy, and M. Takahashi, “Tree Adjunct Grammars”,. *Journal of Computer and System Sciences*, 10 (1), pp. 136-163, 1975.

[9] A.K. Joshi and Y. Schabes, “Tree Adjoining Grammars”, in: *Handbook of Formal Languages*, Rozenberg G. and Saloma A. (Eds.) Springer-Verlag, pp. 69-123, 1997.

[10] J. Koza, *Genetic Programming*,The MIT Press, (1992).

[11] Nguyen Xuan Hoai and R.I. McKay, “A Framework for Tree Adjunct Grammar Guided Genetic Programming”, in: *Proceedings of Post Graduate ADFA Conference on Computer Science (PACCS'01)*, H.A. Abbass and M. Barlow (Eds), pp. 93-99, 2001.

[12] Nguyen Xuan Hoai, R.I., McKay, D. Essam, and R. Chau, “Solving the Symbolic Regression Problem with Tree Adjunct Grammar Guided Genetic Programming: The Comparative Result”, In *Proceedings of Congress on Evolutionary Computation (CEC'2002)*, 2002, pp. 1326-1331.

[13] Nguyen Xuan Hoai, R.I. McKay, and H.A. Abbass, “Tree Adjoining Grammars, Language Bias, and Genetic Programming”, in *Proceedings of EuroGP 2003*, Ryan C. et al (Eds), LNCS 2610, Springer Verlag, 2003, pp. 335-344.

[14] Nguyen Xuan Hoai, R.I. McKay, D. Essam, and H.A. Abbass, ” Toward an Alternative Comparison between Different Genetic Programming System”. To Appear in the *Proceedings of EuroGP'2004*.

[15] Nguyen Xuan Hoai and R.I. McKay, “Softening the Structural Difficulty with TAG-based Representation and Insertion/Deletion Operators”. Submitted paper.

[16] M. O'Neil M. and C. Ryan, “Grammatical Evolution”, *IEEE Trans on Evolutionary Computation*, 4 (4), pp. 349-357, 2000.

[17] Y. Schabes and S. Shieber: An Alternative Conception of Tree-Adjoining Derivation. *Computational Linguistics*, 20(1), pp. 91-124, 1994.

[18] Y. Schabes Y. and R.C. Waters, “Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar

without Changing the Trees Produced", *Computational Linguistics*, 21 (4), pp. 479-514, 1995.

[19] Y. Schabes, *Mathemantical and Computational Aspects of Lexicalized Grammars*, Ph.D. Thesis, University of Pennsylvania, USA, 1990.

[20] V. Shanker, *A Study of Tree Adjoining Grammars,* PhD. Thesis, University of Pennsylvania, USA, 1987.

[21] D.J. Weir, *Characterizing Mildly Context-Sensitive Grammar Formalism,* PhD. Thesis, University of Pennsylvania, USA, (1988).

[22] P.A. Whigham, *Grammatical Bias for Evolutionary Learning*, Ph.D Thesis, University of New South Wales, Australia, (1996).

[23] P.A. Whigham, "Grammatically-based Genetic Programming", in: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Morgan Kaufmann Pub, 1995, pp. 33-41.

[24] M.L. Wong and K.S. Leung, " Evolutionary Program Induction Directed By Logic Grammars", *Evolutionary Computation*, 5, pp. 143-180, 1997.