Softening the Structural Difficulty in Genetic Programming with TAG-Based Representation and Insertion/Deletion Operators

Nguyen Xuan Hoai and R.I. McKay

School of IT & EE, Australian Defence Force academy, University of New South Wales, ACT 2600, Australia.
x.nguyen@adfa.edu.au; rim@cs.adfa.edu.au

Abstract. In a series of papers [3-8], Daida et. al. highlighted the difficulties posed to Genetic Programming (GP) by the complexity of the structural search space, and attributed the problem to the expression tree representation in GP. In this paper, we show how to transform a fixed-arity expression tree in GP to a non fixed-arity tree (Catalan tree) using representation based on Tree Adjoining Grammars (TAGs). This non fixed-arity property, which is called feasibility, allows us to design many types of genetic operators (as in [16]). In particular, insertion/deletion operators arising naturally from the representation play a role as structural mutation operators. By using these dual operators on TAG-based representation, we demonstrate how these operators can help to soften the structural search difficulties in GP.

1 Introduction

Since its original proposal [1, 13], standard genetic programming (GP) has been using expression tree as the representation for programs. In an expression tree, each node has a fixed arity (fixed number of children). We argue that this fixed-arity property makes it hard to design operators that can act on structure of the expression tree. Moreover, in a series of papers [3-8], Daida et. al. have shown that structure search alone can pose a great difficulty to standard GP (using expression tree representation and sub-tree swapping as crossover operator). In particular, they pointed out that vast majority of expression tree structures are essentially not searchable with GP, attributing the problem to the expression tree representation itself.

In this paper, we show how to transform a fixed-arity expression tree in GP to a non fixed-arity tree (Catalan tree) using a representation based on Tree Adjoining Grammars (TAGs). This non fixed-arity property, which we call feasibility, allows us to design many types of genetic operators (as in [16]). In particular, insertion/deletion operators arising naturally from the representation play a role as structural mutation operators. By using these dual operators on the TAG-based representation, we demonstrate how these operators can help to soften the structural difficulty in GP.

K. Deb et al. (Eds.): GECCO 2004, LNCS 3103, pp. 605–616, 2004.

The paper, therefore, proceeds as follows. In section 2, we introduce the idea of TAG-based representation, whereby tree adjoining grammar derivation trees are used as genotypes and the expression trees are phenotypes. We also show how this genotype-to-phenotype map transforms fixed-arity trees to non-fixed-arity trees and obtain deletion and insertion as structural mutation operators in a natural way. Section 3 contains a brief summary of structural difficulty in standard GP, based on work [3-8] by Daida et. al. In section 4, experimental results using insertion/deletion in a hill-climbing search on Daida's LID problem [8] are given; they are discussed, and compared with results of standard GP in [8]. We give a brief survey of related work in section 5 and conclude the paper with section 6, containing some ideas for extending the work.

2 TAG-Based Representation for GP

In this section, we first give the definitions of tree adjoining grammars (TAGs) and their derivation trees. Next, we describe how TAG-derivation trees can be used for genetic programming as in [14-16]. Finally, we describe insertion and deletion operators used with TAG-based representation.

2.1 Tree Adjoining Grammars

Joshi and his colleagues in [11] 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 [12].

TAGs are tree-rewriting systems, defined in [12] 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 ($T \cap V = \varnothing$); $S \in V$ is a distinguished symbol called the start symbol; and $E = I \cup A$ 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 [12] 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 $\downarrow$ for substitution. Initial and auxiliary trees are denoted α and β 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 γ from an auxiliary tree β and a tree α (initial, auxiliary or derived). If tree α has an interior node labeled A, and β is an A-type tree, the adjunction of β into α to produce γ is as follows: Firstly, the sub-tree α_1 rooted at A is temporarily disconnected from α. Next, β is attached to α to

replace the sub-tree. Finally, α_1 is attached back to the foot node of β. γ is the final derived tree achieved from this process. Adjunction is illustrated in Figure 2.

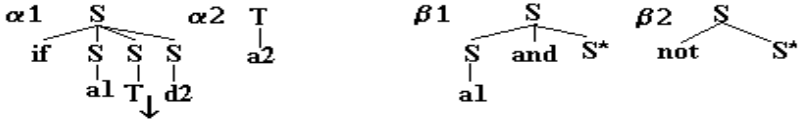

Fig. 1. Some examples of initial and auxiliary trees.

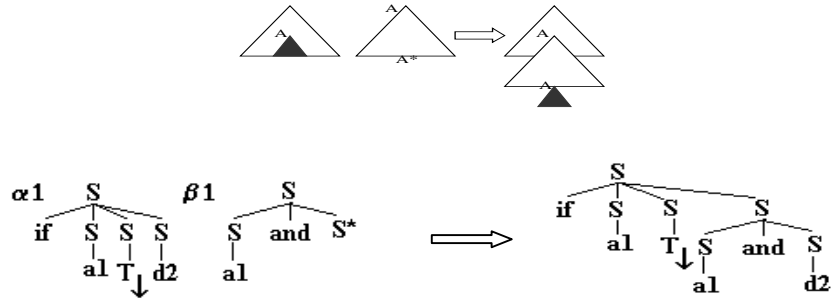

Fig. 2. Adjunction.

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 [12]:

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 [12, 19, 23, 26] 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 α tree name, and the other nodes with either an α 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).

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 [12]. More properties of TAL can be found in [12].

One special class of TAGs is lexicalized TAGs (LTAGs) [12, 20, 21], 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) [12, 20, 21]. The derivation trees in G are the derived trees of G_{lex}.

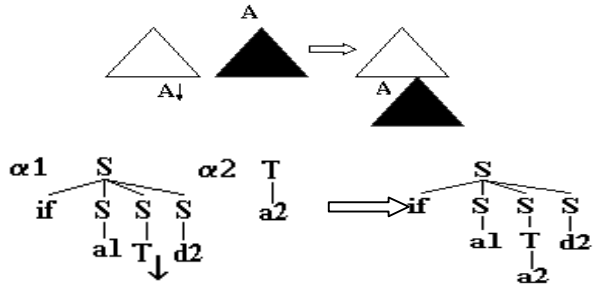

Fig. 3. Substitution.

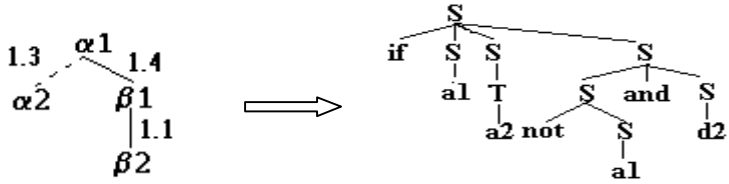

Fig. 4. Examples of a derivation tree and derived tree in TAGs.

2.2 TAG-Based Representation for Genetic Programming

The algorithm in [12, 20, 21] 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 [21], the only operation necessary in the resultant LTAG is adjunction. However, substitution can be added to make the elementary set more compact [12]. 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).

In [16], the derivation tree in LTAG was used as genotype structure for Tree-Adjoining Grammar-Guided Genetic Programming (TAG3P). 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 trees 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 a set of GP functions and terminals is used to create a context-free

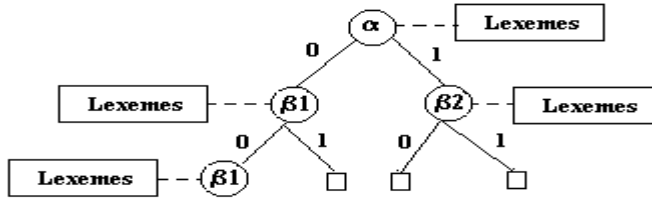

Fig. 5. Derivation tree structure for TAG-based representation. The squares means there is no tree adjoining to that address (a NULL node). Lexemes are values for the lexicons in each node.

grammar in the manner described in [29] (page 130). It was proven in [29] 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.

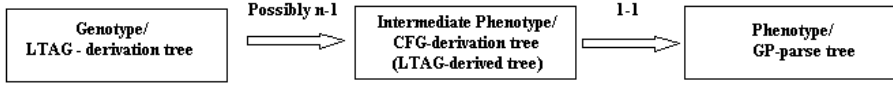

Fig. 6. Scheme for Genotype-to-Phenotype map in TAG-based Representation.

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 [22]). 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 [16], this property was called feasibility. Feasibility allows us to design and implement many new search operators which would not be possible in standard GP systems, including bio-inspired ones. 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 replaced 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.

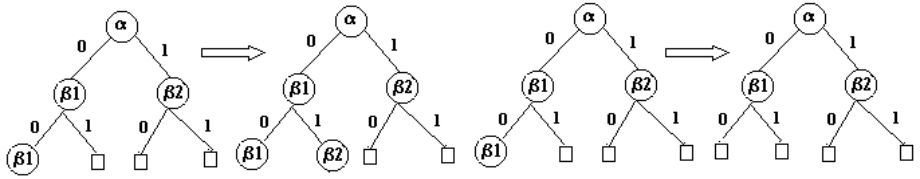

Fig. 7. Examples of insertion (on the left) and deletion (on the right).

3 Structural Difficulty in GP

In a series of works [3-8], Daida et. al. showed that structure alone can pose great difficulty to standard GP search (using expression tree representation and sub-tree swapping crossover). In particular, they delineated 4 regions of the space tree structure [6, figure 3]. Region 1 is where most of standard GP solutions lie, i.e. it is easy for GP to find a solution in that area; region 2 is increasingly difficult for GP to search; region 3, including fuller trees and thinner trees, is almost impossible for GP to search; and, region 4 is out of bounds (i.e. infeasible tree structures). Moreover regions 2 and 3 account for the majority of tree structures, even when, as is usual in practical GP, a relatively small search space bound (in terms of size or depth) is used (see [6, figure 3]).

To further validate this discovery, in their latest paper [8], Daida et al. specified a test problem known as LID. In the LID problem for GP, there is only one function of arity 2 named join, and one terminal named leaf. The raw fitness of one individual tr depends purely on its structural difference from the target solution. It is defined as follows [8].

$$\text{Fitness}_{raw}(tr) = \text{Metric}_{depth} + \text{Metric}_{terminal} \quad (1)$$

Where Metric_{depth} and $\text{Metric}_{terminal}$ are defined as:

$$\text{Metric}_{depth} = W_{depth} \times \left(1 - \left(\frac{|d_{target} - d_{actual}|}{d_{target}}\right)\right) \quad (2)$$

$$\text{Metric}_{terminal} = \begin{cases} W_{terminal} \times \left(1 - \left(\frac{|t_{target} - t_{actual}|}{t_{target}}\right)\right) & \text{if } \text{Metric}_{depth} = W_{depth} \\ 0, \text{ otherwise} \end{cases} \quad (3)$$

d_{target}, and t_{target} are the depth and number of leaves of the target solution, and d_{actual} and t_{actual} are the depth and number of leaves of individual tr. In [8], W_{depth} and $W_{terminal}$ are two weighted numbers satisfying $W_{depth} + W_{terminal}=100$. It is noted that the size s of a tree in LID problem is related to its t_{target} by the equation: $s=2\times t_{target}-1$.

In [8], two families of LID problem instances were used to probe the search space of tree structure: 'horizontal cut' and 'vertical cut'. In the first family, the t_{target} was fixed as 256 and the d_{target} was varied from 8 to 255. In the second, d_{target} was fixed as 15 while t_{target} was varied from 16 to 32768. For a GP system using either size or depth as the chromosome complexity measure, these bounds on size and depth (256 and 15) are quite typical.

Surprisingly, the results in [8, figure 3 and 4] show that standard GP, using expression tree representation and sub-tree swapping crossover, performed extremely poorly on the two families of problem instances, especially for those vertical and horizontal cut regions that lie in regions 3 and 4. The results thus support the hypothesis that there is great structural difficulty in GP. Daida et. Al. [8, figures 6 and 7] went further, in showing that the results cannot be fully explained by the sparsity of tree structures in regions 3 and 4 (i.e. they are not an equilibrium problem). Their explanation pointed to the expression tree representation itself as the main cause of the structural difficulty.

Our further work on this problem has identified the lack of appropriate structural edit operators, resulting from the fixed-arity expression tree representation in standard GP, as the culprit. Using TAG-based representation we have been able to solve the problem of fixed-arity and design structural mutation operators, namely, insertion and deletion. In the next section, we investigate how the TAG-based representation, coupled with insertion/deletion operators, softens the structural difficulty in GP.

4 Experiments and Results

To investigate how well TAG-based representation and insertion/deletion operators handle the structural difficulty, we tried a hill-climbing (greedy) search using insertion/deletion as operators (TAG-HILL) on the same families of LID problem instances as in [8], namely the horizontal cut and the vertical cuts. The grammar for the LID problem is as follow:

G={N={S}, T={Join, Leaf },P,{S}} where the rule set P is defined as follows:

$S \rightarrow S$ Join S

$S \rightarrow$ Leaf

The corresponding LTAG (as found in [21]) is G_{lex}= {V={S}, T={Join, Leaf}, I, A) where I$\cup$ A is as in Figure 8.

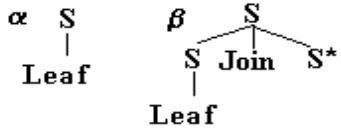

Fig. 8. Elementary trees of G_{lex} for LID problem.

It is noted in [12] that the map between derivation tree of G_{lex} and derivation tree of G (and therefore to expression tree of GP) is one-to-one. However, while the GP expression tree for LID is a fixed-arity (binary) tree, the derivation tree in G_{lex} is a non-fixed-arity (Catalan binary) tree.

In our experiments, the maximal number of steps in TAG-HILL is set to 100000 for each instance. This gives the same number of evaluations as in [8], where the size of population and the number of generations are set to 500 and 200 respectively. Consequently, the maximal allowed number of fitness evaluation in TAG-HILL is the same as in the GP experiments in [8]. For the horizontal cut, t_{target} was fixed as 256 while d_{target} was varied from 8 to 255. For each varied d_{target}, 100 runs were allocated. Similarly, in the vertical cut, the d_{target} was fixed as 15 while t_{arget} was varied from 16 to 32768. For each t_{target} in [16..1024], we carried out 100 runs, while in [1025..32768], we run 100 trials for each point of the 118 equi-sampled points in that interval. Although the setting of W_{depth} and $W_{terminal}$ do not affect TAG-HILL search, we still set them the same as in [8], i.e. as 30 and 70 respectively. The size of the initial individuals is randomly chosen between 2 and 10. Both operators have equal chance of being chosen. Figures 9 and 10 depict the results on the frequency of success for TAG-HILL compared with the GP results reported in [8]. The results for TAG-HILL

are based on total 137600 runs. The graphs of GP results are adapted from figures 3 and 4 in [8] (based on 90000 runs).

The results show that TAG-HILL outperforms GP on the two families of LID problem instances by an extremely wide margin. For the horizontal cut family, TAG-HILL solved all except the three rightmost points (where the frequencies of success were 98%, 80%, 72% respectively) with 100% reliability, while GP could not reliably find solutions for the whole range of problems with $d_{target} < 12$ and $d_{target} > 70$; with $d_{target} > 100$ and $d_{target} = 8$ or 9, GP failed to find any solutions. For the vertical cut family, GP runs were unreliable in finding solutions for $t_{target} > 500$, and failed to find any solutions with $t_{target} > 1024$. By contrast, TAG-HILL can solve the problem with 100% success for t_{target} up to 13400, and only failed to find solutions when $t_{target} > 16000$. Figures 11 and 12 show the average number of search steps for TAG-HILL to find solutions. The almost linear scale suggests that, except for some extreme points, the landscape of the two families of LID problem instances is quite smooth for TAG-HILL. This is no trivial matter, since when t_{target} (d_{target}) approach their extreme values, the tree structure become exponentially sparse [23]. To see just how sparse, take the example of the leftmost point on the horizontal cut where $t_{target} = 256$ and $d_{target} = 8$. There is only one tree with that combination of (t_{target}, d_{target}) out of $\frac{4^{255}}{\sqrt{\pi 255^3}} \sim 2^{497}$ trees with t_{target} of 256 [23].

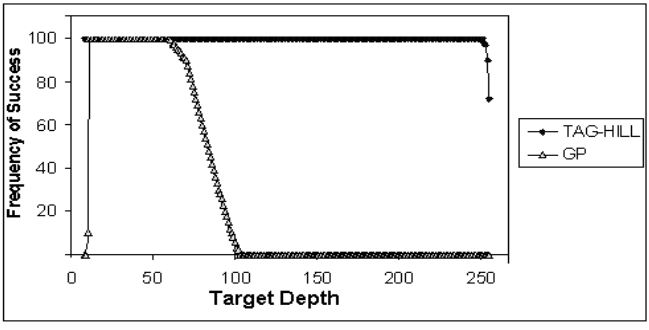

Fig. 9. Frequency of success for the 'horizontal cut'.

The results show that TAG-based representation, equipped with insertion/deletion, can soften the structural difficulties in GP. Of course, except for specially tailored problems (like LID) it may be difficult to determine whether a problem has inherent structural difficulty and hence whether to apply insertion and deletion. Further investigation of this issue will be deferred to our future work. In other recent work [17], we have investigated the role of insertion and deletion as structural mutation and/or local search operators in the context of Tree Adjoining Grammar Guided Genetic Programming (TAG3P). The results show that, on some standard GP problems, insertion and deletion help TAG3P to improve the performance significantly, even with very small population sizes.

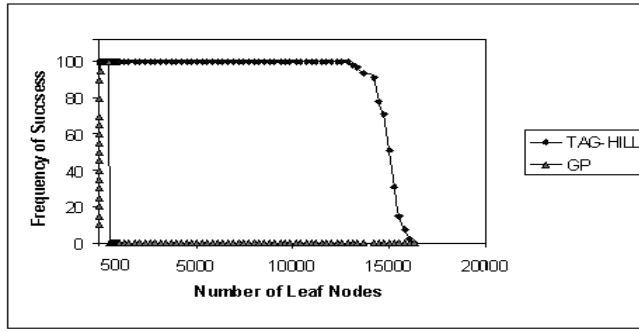

Fig. 10. Frequency of success for the 'Vertical cut'.

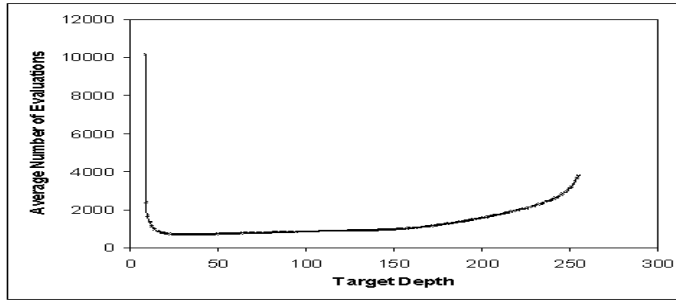

Fig. 11. Average number of Evaluations (TAG-HILL) for the 'horizontal cut'.

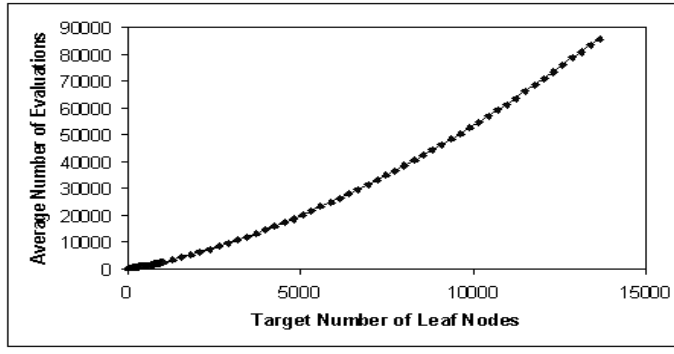

Fig. 12. Average number of Evaluations (TAG-HILL) for the 'vertical cut'.

5 Related Works

To the best of our knowledge, in the field of grammar guided genetic programming (GGGP) [9, 10, 18, 27-30], structural mutation and the problem of structural difficulty have not yet been studied. We believe that the structural difficulty problem also

affects the derivation tree structure of (context-free, logic) grammars (at least when solving non-syntactical constraints like in GP). However, it would be extremely difficult in GGGP to solve the problem using structural mutations like our TAG-based insertion and deletion, because of the complexity of defining them through syntactical (string-rewriting) rules.

Recently, Vanneschi et al. [24, 25] designed new GP structural mutation operators, called inflate mutation and deflate mutation, for studying fitness-distance correlation. They formally proved that their structural mutation operators are consistent with the alignment-tree metric. This means, if the alignment metric between to tree t_1 and t_2 is D, there is a sequence of inflate and deflate mutations with length D/2 to transform t1 into t2. However, it is not yet known whether this optimal sequence can be obtained in real time. Being based on the arity incremental ranking of nodes, their operators become meaningless when all the functions in GP have the same arity (as in LID), and it is hard to imagine how these operators can be extended to handling syntactically constrained domains. Moreover, their structural mutation operators are complicated in implementation and unnatural in definition.

6 Conclusion

In this paper, we have reintroduced a representation for genetic programming, based on the use of tree adjoining grammars. We described in detail how this representation transforms the fixed-arity expression tree representation in GP into a non-fixed-arity tree structure in LTAG. From this property of TAG-based representation (it was named feasibility in [16]), insertion and deletion arise naturally as structural mutation operators. The results on two families of LID problem instances, using stochastic hill-climbing search, show that the TAG-based representation significantly softens the structural difficulty previously found in standard GP using expression tree representation and sub-tree swapping crossover [8].

In future, we are planning to analyze the run time behaviour of TAG-HILL and GP (using sub-stree swapping crossover) on the LID problem, aiming at prediction of the convergence time. We are developing a way to measure the structural difficulty of problems, aiming to predict the usefulness of insertion and deletion for those problems.

References

1. Banzhaf W., Nordin P., Keller R.E., and Francone F.D.: *Genetic Programming: An Introduction*. Morgan Kaufmann Pub (1998).
2. Daida J.M., Ross S.J., McClain J.J., Ampy D.S., and Holczer M.: Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Koza J. etal.(Eds), Morgan Kaufmann, (1997) 64-69.
3. Daida J.M., Bertram J.A., Polito 2 J.A., and Stanhope S.A.: Analysis of Single-Node (Building) Blocks in Genetic Programming. In *Advances in Genetic Programming 3*, Spector L., Langdon W.B., O'Reilly, and Angeline P.J. (Eds), the MIT Press (1999) 217-241.

4. Chaudri O.A., et al.: Characterizing a Tunably Difficult Problem in Genetic Programming. In *Proceedings of GECCO 2000*, Witley L.D., et al. (Eds), Morgan Kaufmann Publisher (2000) 395-402.
5. Daida J.M., Polito 2 J.A., Stanhope S.A., Bertram R.R., Khoo, J.C., Chaudhary S.A., and Chaudhri O.: What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming. *Journal of Genetic Programming and Evolvable Machines*, vol. 2 (2001) 165-191.
6. Daida J.M.: Limit to Expression in Genetic Programming: Lattice-Aggregate Modeling. In *Proceedings of the 2002 Congress on Evolutionary Computation*, IEEE Press (2002) 273-278.
7. Daida J.M. and Hilss. Identifying Structural Mechanism in Standard GP. In *Proceedings of GECCO 2003*, LNCS, Springer-Verlag (2003) 1639-1651.
8. Daida J.M., Li H., Tang R., and Hilss A.M.: What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes. In *Proceedings of GECCO 2003,* LNCS, Springer-Verlag (2003) 1665-1677.
9. Gruau F.: On Using Syntactic Constraints with Genetic Programming. In: *Advances in Genetic Programming* II, The MIT Press, (1996) 377-394.
10. Geyer-Schulz A.: *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*. Physica-Verlag, Germany, (1995).
11. Joshi A. K., Levy L. S., and Takahashi M.: Tree Adjunct Grammars. *Journal of Computer and System Sciences*, 10 (1), (1975) 136-163.
12. Joshi, A. K. and Schabes, Y.: Tree Adjoining Grammars. In: *Handbook of Formal Languages*, Rozenberg G. and Saloma A. (Eds.) Springer-Verlag, (1997) 69-123.
13. Koza, J. : *Genetic Programming*. The MIT Press (1992).
14. Nguyen Xuan Hoai and McKay R.I.: 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), (2001) 93-99.
15. Nguyen Xuan Hoai, McKay R.I., Essam D., and Chau R.: 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) 1326-1331.
16. Nguyen Xuan Hoai, McKay R.I., and Abbass H.A.: Tree Adjoining Grammars, Language Bias, and Genetic Programming. In *Proceedings of EuroGP 2003*, Ryan C. et al (Eds), LNCS 2610, Springer Verlag (2003) 335-344.
17. Nguyen Xuan Hoai and McKay R.I: An Investigation on the Roles of Insertion and Deletion Operators in Tree Adjoining Grammar Guided Genetic Programming. To appear in *The Proceedings of Congress on Evolutionary Computation* (CEC'2004) (2004).
18. O'Neil M. and Ryan C.: Grammatical Evolution. *IEEE Trans on Evolutionary Computation*, 4 (4), (2000) 349-357.
19. Schabes Y. and Shieber S.: An Alternative Conception of Tree-Adjoining Derivation. *Computational Linguistics*, 20 (1), (1994) 91-124.
20. Schabes Y. and Waters R. C.: Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced. *Computational Linguistics*, 21 (4), (1995) 479-514.
21. Schabes Y.: *Mathemantical and Computational Aspects of Lexicalized Grammars*, Ph.D. Thesis, University of Pennsylvania, USA, (1990).
22. Sedgewick R. and Flajolet P: *An Introduction to the Analysis of Algorithms*. Addison-Wesley (1996).
23. Shanker V.: *A Study of Tree Adjoining Grammars*. PhD. Thesis, University of Pennsylvania, USA, 1987.

24. Vanneschi L., Tomassini M, Collard P., and Clergue M: Fitness Distance Correlation in Structural Mutation Genetic Programming. In *Proceedings of EuroGP 2003*, Ryan C. et al (Eds), LNCS 2610, Springer Verlag, (2003) 455-464.
25. Vanneschi L., Tomassini M, Collard P., and Clergue M: Fitness Distance Correlation in Genetic Programming: a Constructive Counterexample In *Proceedings of Congress on Evolutionary Computation (CEC'2003)*, IEEE Press (2003) 289-296.
26. Weir D. J.: *Characterizing Mildly Context-Sensitive Grammar Formalisms.* PhD. Thesis, University of Pennsylvania, USA, (1988).
27. Whigham P. A.: *Grammatical Bias for Evolutionary Learning*. Ph.D Thesis, University of New South Wales, Australia, (1996).
28. Whigham P. A.: Grammatically-based Genetic Programming. In: *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Morgan Kaufmann Pub (1995) 33-41.
29. Whigham P. A.: Search Bias, Language Bias and Genetic Programming. In: *Genetic Programming* 1996, The MIT Press, USA, (1996) 230-237.
30. Wong M. L. and Leung K. S.: Evolutionary Program Induction Directed by Logic Grammars. *Evolutionary Computation*, 5 (1997) 143-180.