Representation and Structural Difficulty in Genetic Programming

Nguyen, Xuan Hoai, *Student Member, IEEE,* R I (Bob) McKay, *Senior Member, IEEE,* and Daryl Essam

***Abstract*— Standard tree-based genetic programming suffers from a structural difficulty problem, in that it is unable to search effectively for solutions requiring very full or very narrow trees. This deficiency has been variously explained as a consequence of restrictions imposed by the tree structure, or as a result of the numerical distribution of tree shapes. We show, that by using a different tree-based representation and local (insertion and deletion) structural modification operators, that this problem can be almost eliminated even with trivial (stochastic hill-climbing) search methods, thus eliminating the above explanations. We argue instead, that structural difficulty is a consequence of the large step size of the operators in standard genetic programming, which is itself a consequence of the fixed-arity property embodied in its representation.**

***Index Terms*— Genetic Programming, Structural Difficulty, representation, operator, insertion, deletion.**

I. Introduction

SINCE its very beginnings in the late 1980s [4], [20], Genetic Programming (GP) has relied on tree representation as a key element. However recent work by Daida and his colleagues [10], has cast doubt on our understanding of tree representation for GP, by showing that the standard representation and operators generate important anomalies. They demonstrated that evolutionary search on this representation is unable to effectively search all tree shapes, and in particular, that very full or very narrow tree solutions may be extraordinarily difficult to find, even when the fitness function provides good guidance to the optimum solution. These results have important ramifications for tree-based GP. They suggest that GP will perform poorly on problems where solutions require full or narrow trees (which we may not even know in advance). Even more worrying, they raise the possibility (since the difficulties arise at both ends of this 'fullness' spectrum) that this problem may arise just from the requirement for a particular tree structure, and hence may apply to any problem whose solutions are restricted to a particular shape, of whatever degree of fullness.

Daida and his colleagues blame this difficulty on the inflexibility of the tree representation, while other explanations have been discussed in the GP community based on the relative sparseness of full and narrow trees (in which case we would not need to be concerned about structural difficulty for problems whose solutions have intermediate levels of fullness, since they are not sparse).

In this work, we propose a new tree-based representation and simple local operators for GP. Using these operators and a very simple search strategy (stochastic hill-climbing), we show that the structural difficulty problem is very largely ameliorated, thus disposing of the two previous explanations of the structural difficulty problem. Namely, that this problem is not due simply to the tree representation, since the new representation is also tree-based. It is also not a simple consequence of the sparseness of particular tree structures, since these are equally sparse under the new representation. We propose a new hypothesis, based on the connectivity of neighbourhood topologies, to explain the structural difficulty problem.

In Section II, we re-visit the structural difficulty problem. We propose a hypothesis for the cause of structural difficulty in tree-based GP. We then introduce Tree Adjoining Grammars (TAGs), which provide the basis for our alternative representation. Section III details the representation and introduces the point insertion and deletion operators which are the key to our approach, discussing their relationship with the distance metric on both genotype and phenotype space. It also details the stochastic hill-climbing search algorithm used in the paper. Details of the experimental regime are given in Section IV, while Section V presents the results of the experiments and discusses their meaning. Finally, in Section VI, we examine the implications of the experiments.

II. Background and Previous Work

In this section, we review the structural difficulty problem in GP and propose a new hypothesis on that problem. The section ends with some basic concepts of tree adjoining grammars.

A. Structural Difficulty in Genetic Programming

In a series of papers [3], [5]–[10], Daida et. al. showed that structure alone can pose great difficulty for standard Genetic Programming (GP) search (using an expression tree representation and sub-tree swapping crossover). In particular, they delineated 4 regions of the search space of tree structures, as shown in Figure 1

Most solutions found by standard GP search lie in region I. To put it another way, it is easy for GP to find solutions to problems which lie in region I. GP has greater difficulty in searching the next region, region II (II_a, II_b). By the time

This is a self-archived copy of the accepted paper, self-archived under IEEE policy. The authoritative, published version can be found at http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1613934&tag=1

Manuscript submitted November 30, 2004.

Revised version submitted August 31, 2005.

This work was undertaken while all authors were with the University of New South Wales at the Australian Defence Force Academy.

Nguyen, Xuan Hoai is now at the Army Technical Academy of VietNam, and Bob McKay is now at Seoul National University, Korea

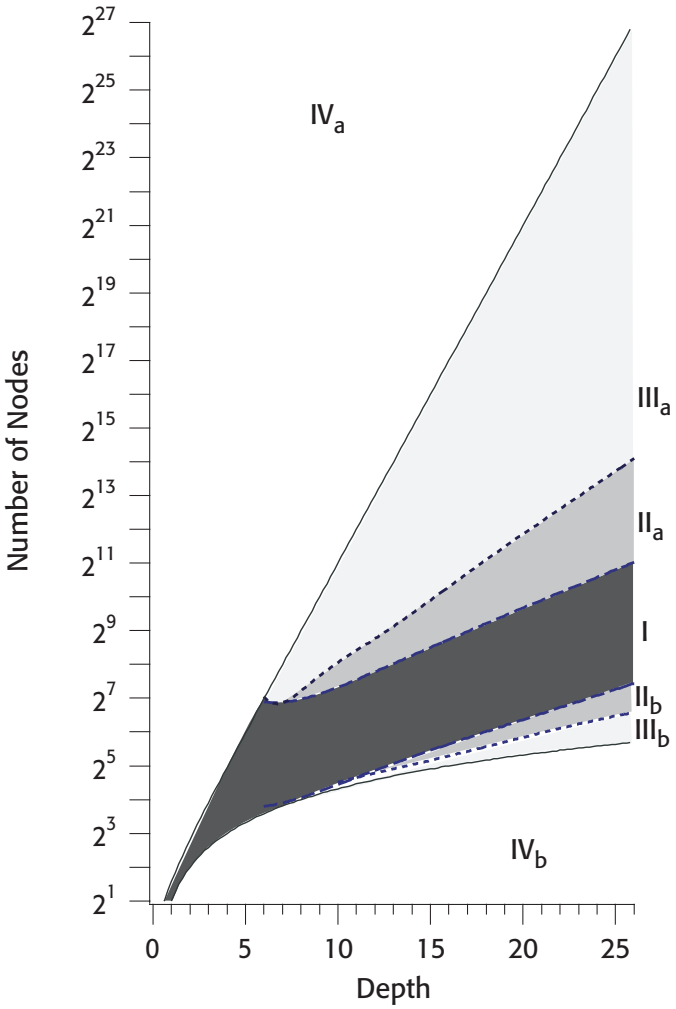

Fig. 1. Four regions in the space of tree structures. Reprinted with permission from [10]

we come to region III (III_a, III_b, respectively very wide and very narrow trees), GP is effectively unable to search in this region, and will not find solutions there. Of course once the ratio of depth to number of nodes becomes too large or too small, it is impossible to construct trees. There are no feasible tree structures in region IV (IV_a, IV_b). Daida and his colleagues noted that the boundaries of regions II and III are almost linear, meaning that they account for the vast majority of tree structures, even when, as is usual in practical applications of GP, a relatively small search space bound is used.

To further validate this analysis, in recent work [10], Daida et al. specified a test problem known as LID. In the LID problem for GP, there is only one arity 2 function, named **join**, and one terminal named **leaf**. The raw fitness of an individual tr depends purely on its structural difference from the target solution. It is defined as follows:

$$Fitness_{raw}(tr) = Metric_{depth} + Metric_{term} \quad (1)$$

Where $Metric_{depth}$ and $Metric_{term}$ are defined as follows:

$$Metric_{depth} = W_{depth} \times (1 - \frac{|d_{target} - d_{actual}|}{d_{target}}) \quad (2)$$

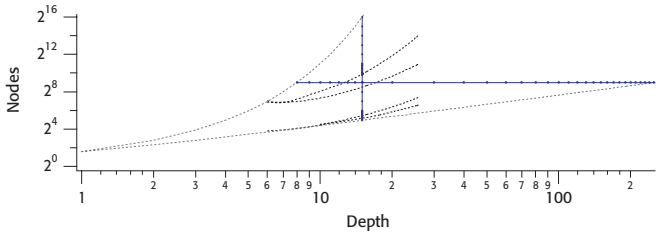

Fig. 2. The 'horizontal and vertical cuts'. Reprinted with permission from [10]

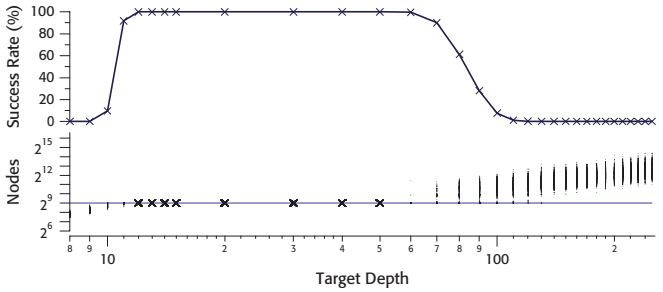

Fig. 3. Proportion of Success for GP on the 'Horizontal cut'. Reprinted with permission from [10]

$$\begin{aligned} Metric_{term} &= W_{term} \times (1 - \frac{|t_{target} - t_{actual}|}{t_{target}}) \quad (3) \\ &\quad \text{if } Metric_{depth} = W_{depth} \\ &= 0, \text{ Otherwise.} \end{aligned}$$

where 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 the individual (tree) tr. In [10], W_{depth} and W_{term} are two weighted numbers satisfying $W_{depth} + W_{term} = 100$. We note that the size s of a tree in the LID problem is related to its t_{target} by the equation: $s = 2 \times t_{target} - 1$.

In [10], two families of LID problem instances were used to investigate the search space of tree structures, namely "horizontal cut" and "vertical cut". In the first family, the t_{target} was fixed at 256 and the d_{target} was varied from 8 to 255. In the second, d_{target} was fixed at 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. Figure 2 is a simplified version of Figure 1 with the positions of the problem instances superimposed on it.

Figures 3 and 4 show the results, based on 90000 runs, of GP (standard representation with subtree crossover) on the two families of problem instances. We note that in Figure 4, the x-axis is the number of target nodes (s), which is approximately twice the value of t_{target}. The upper parts of the figures show the proportion of successful runs, while the lower show the region to which each problem instance belongs (the cross sign means region I, while the vertical line means regions II and III).

The results in the two figures, surprisingly, show that

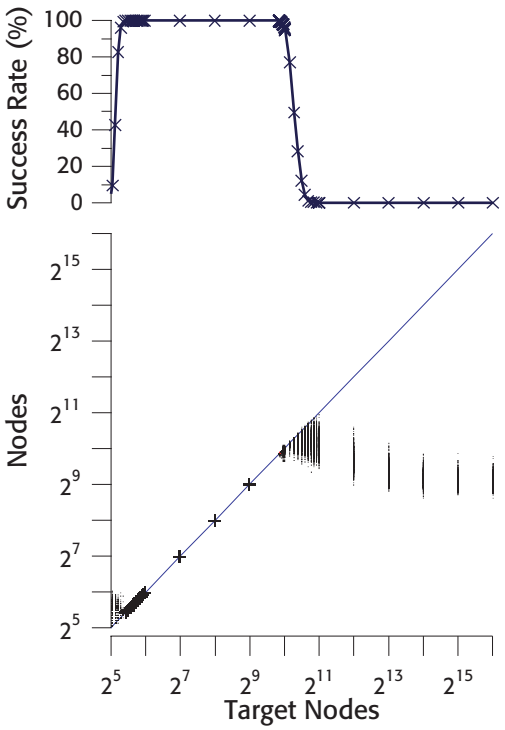

Fig. 4. Proportion of Success for GP on the 'Vertical cut'. Reprinted with permission from [10]

standard GP, using expression tree representation and sub-tree-swapping crossover, performed extremely poorly on the two families of problem instances, especially for vertical and horizontal cut problems lying in regions II and III. This provides strong evidence that standard GP has considerable difficulty in finding specific structures. Daida et al. [10] went further, in showing that these results cannot be fully explained by the sparsity of tree structures in regions II and III (i.e. they are not an equilibrium problem). Their explanation conjectured that the expression tree representation was itself the main cause of the structural difficulty.

1) Structural Difficulty and Operator Step Size: Elaborating on Daida's explanation, we conjecture that in standard GP, the problem lies in the structural step size of the structure editing operators. In other words, sub-tree crossover and sub-tree mutation, the two main operators in GP, are highly structurally discontinuous. Hence, despite the presence of selection pressure, the probability of exploring regions 2 and 3 in the space of tree structures is low. We further argue that this discontinuity is a consequence of the fixed-arity property of standard GP representation (that is, each node in the tree has a fixed number of children, determined by its content), in that fixed arity makes it difficult to design operators with a controllable step size.

There are at least two ways to validate this hypothesis. One is to analytically analyze the probability of reaching regions 2 and 3 using the particular operator set; the other is to design structure editing operators and show that they can help GP to solve the problem of structural difficulty. We adopt the second course in this paper, making use of a formalism derived from natural language processing, Tree Adjoining Grammars.

B. Tree Adjoining Grammars

Tree adjoining grammars (TAGs) are tree-generating and analysis systems, first proposed by Joshi et al in [14]. Tree Adjoining Grammars (TAGs) have become increasingly important in Natural Language Processing (NLP) since their introduction.

The aim of TAGs is to more directly represent the structure of natural languages than is possible in Chomsky languages, and in particular, to represent the process by which natural language sentences can be built up from a relatively small set of basic linguistic units by the inclusion of insertable sub-structures. Thus 'The cat sat on the mat' becomes 'The big black cat sat lazily on the comfortable mat which it had commandeered' by the subsequent insertion of the elements 'big', 'black', 'lazily', 'comfortable', and 'which it had commandeered'. In context-free grammars (CFG)(Chomsky's formalisms of type 2), the relationship between these two sentences can only be discerned by detailed analysis of their derivation trees; in a TAG representation, the derivation tree of the latter simply extends the frontier of the former. To put it another way, the edit distance between the derivation trees of these closely related sentences, is much smaller in a TAG representation than in a CFG representation.

At first, the only tree rewriting operation in TAGs was adjunction (described below) and the formalisms were called tree adjunct grammars. After a sequence of developments in [15]–[18], a new operation, called substitution (described below), was added and the formalisms have subsequently been known as tree adjoining grammars. Another operation, substitution, does not change the family of languages that can be represented by TAG grammars, and in that sense is a redundant element. However, substitution often dramatically reduces the grammar complexity required to represent typical (natural and computer) languages, and hence is important for practical application.

From their early days, TAGs (tree adjunct grammars and tree adjoining grammars) were shown to possess a number of invaluable properties for handling various issues in natural language processing [16], [21], [22]. Some of these have natural analogues in GP. A comprehensive overview of TAGs can be found in [19].

Definition 2.1 (Tree Adjoining Grammars)
A tree adjoining grammar is a tree-rewriting system consisting of a quintuple $(\sum, N, I, A, S)$, where:

1) $\sum$ is a finite set of terminal symbols.

2) N is a finite set of non-terminal symbols: $N \cap \sum = \emptyset$.

3) S is a distinguished non-terminal symbol: $S \in N$.

4) I is a finite set of finite trees, called initial trees (or α-trees).
In an initial tree, all interior nodes are labeled by non-terminal symbols, while the nodes on the frontier are labeled either by terminal or non-terminal symbols. Non-terminal symbols on the frontier of an initial tree

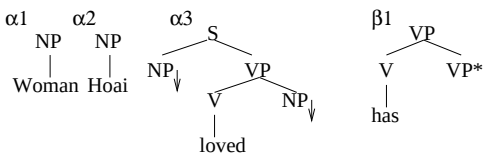

Fig. 5. A simple TAG for some English sentences

are marked with ↓ (for substitution).

5) A is a finite set of finite trees, called auxiliary trees (or β-trees).
In an auxiliary tree, all internal nodes are labeled by non-terminal symbols, and a node on the frontier is labeled either by a terminal or non-terminal symbol. The frontier must contain a unique, distinguished node, the foot node, labeled by the same non-terminal symbol as the tree's root node, and marked with an asterisk (*); other nodes on the frontier labeled by non-terminal symbols are marked with ↓ (for substitution).

The trees in $E = I \cup A$ are called elementary trees. Initial trees and auxiliary trees are denoted as α and β trees respectively. A tree with its root labeled by a non-terminal symbol X is called an X-type elementary tree.

In essence, an α-tree which has terminal symbols on its frontier, is just like a minimal complete sentence, while a β-tree is a minimal recursive structure which may be used to modify complete sentences (by using adjunction as described below).

TAG Example. G_1=$\{\sum, N, I, A, S\}$, where $\sum$ is a set of English words, $N = \{S, VP, NP, V\}$ and $E = I \cap A$ is given in Figure 5.

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 another tree α. If α has an interior node labeled A, and β is an A-type tree, the adjunction of β into α produces γ as follows: Firstly, the sub-tree α_1 rooted at A is temporarily disconnected from α. Next, β is attached 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 6. Finally, for reference, a node labeled A in an elementary tree is called an adjoining address, if there is an A-type $beta$ tree (i.e if there is a $beta$ tree that can adjoin to that address).

In substitution, a non-terminal node on the frontier of an elementary tree is substituted by an initial tree whose root is labeled with the same non-terminal. Substitution is illustrated in Figure 7.

The addition of substitution does not change the class of languages defined by TAG, but it helps to make the formalism more compact by reducing the size of the elementary tree set.

A special class of TAGs known as lexicalised TAGs (LTAGs) can be defined as follows [19]

Definition 2.4 (Lexicalised TAGs)
A lexicalised tree adjoining grammar (LTAG) is a tree adjoining grammar (TAG) satisfying the requirement that each of its

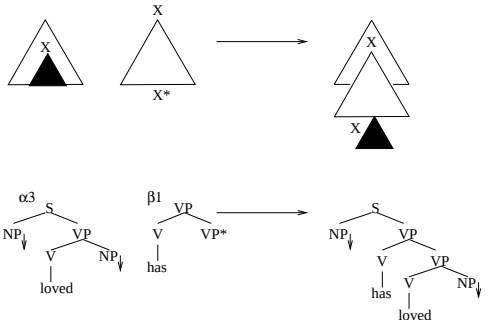

Fig. 6. Adjunction Operation

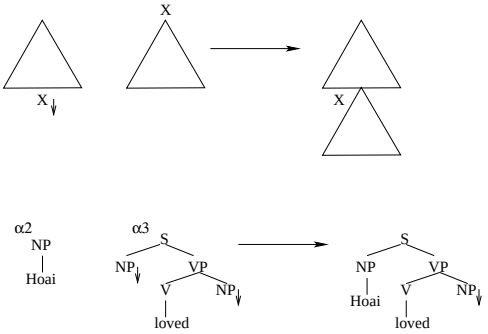

Fig. 7. Substitution

elementary trees contains a terminal node.

Although there are more constraints on LTAGs, it has been shown that LTAGs are equivalent to TAGs (i.e. capable of generating the same languages). In the remainder of this paper, we will deal only with LTAGs, and use the terminology, tree adjoining grammar, interchangeably to also refer to LTAGs, denoting an LTAG with the notation G_{lex}.

C. Derivation Trees in Tree Adjoining Grammars

In TAGs, there is a distinction between derivation and derived trees, where the former encodes the sequence of adjunctions and substitutions used to generate the latter. There are a number of definitions of TAG derivation trees in the literature [19], [37], [42], [47]; the variant we use is described below.

A TAG derivation tree is a labeled object tree satisfying the following requirements. The root node is labeled with the name of an S-type initial ($alpha$) tree; the nodes other than the root are labeled with names of auxiliary (β) trees. Each link between a parent and a child node is labeled with an index, indicating the location in the elementary tree of the parent node to which the auxiliary tree in the child node is to be adjoined. At most one adjunction is permitted at each location. Each node also has attached a list of initial trees (lexemes) to be substituted into open (i.e. unadjoined) locations (lexicons). The corresponding tree generated by performing the specified process of adjunction and substitution is known as the derived tree. Figure 8 shows the structure of this formulation of TAG derivation and derived trees. Note that the derivation trees of a CFG correspond to the derived trees of a TAG.

The set of derived trees which may be generated by a TAG is known as its tree language, while the set of strings which

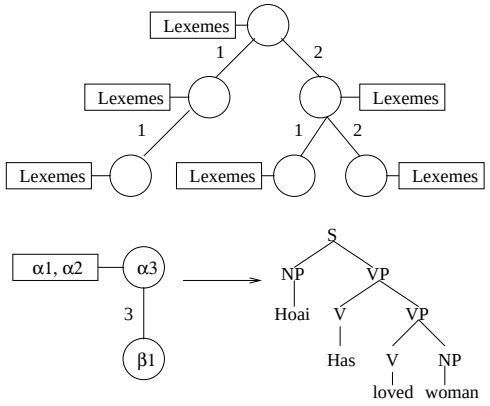

Fig. 8. TAG derivation tree and corresponding derived tree

may be generated from these derived trees is known as its string language.

In this form of derivation tree, substitution is left to last (i.e. all adjunctions are carried out before any substitutions). To simplify the figures in the rest of this paper, substitution will be omitted from the figures, as it is unimportant for the grammars used.

1) Some Properties of TAGs: TAG derivation trees, as described above, have an important property. It is possible to remove any sub-tree, and the resultant tree is still a perfectly valid TAG derivation tree, and its derived tree is still a completed tree. In other words, unlike GP structure trees or CFG derivation trees, the arity (number of children) of each node in a TAG derivation tree is not fixed. This property is crucial to what follows in this paper. Apart from this non-fixed-arity property, TAGs have a number of other important properties:

- The TAG string languages strictly include CFG languages and are strictly included in indexed languages.
- The set of CFG derivation tree sets is strictly included in the TAG tree languages.
- For every context-free grammar G, there is an algorithmic derivation of an LTAG G_{lex} whose tree language is the set of G derivation trees. G_{lex} is said to strongly lexicalise G. Informally, context-free grammars may be algorithmically converted to TAGs [19], [34]–[36], [38].

Thus, the TAG representation is sufficiently powerful to cover the range of search spaces used in grammar guided genetic programming (GGGP) [48], and hence also standard GP.

III. Methodology

In this section, we introduce the TAG representation, operators and search algorithm we used to investigate the problem of structural difficulty.

A. Representation and Operators

1) TAG Representation for Genetic Programming: As a problem representation, we used the LTAG derivation trees defined in the previous section. Thus, the domain of the problem may be delineated by an LTAG grammar, G_{lex}. Since LTAGs can generate the tree set of context-free languages (and some context-sensitive languages), the TAG-based representation can be applied to any problems whose description languages are context-free, as well as to some context-sensitive problems.

TABLE I
PSEUDOCODE FOR INITIALISATION PROCEDURE

```
1)   FOR i = 1 TO POPSIZE DO
2)   Choose a a random size l between
       MINSIZE and MAXSIZE.
3)    Pick an α-tree α_1 at random
       and set tree T = α_1.
4)    FOR j = 1 TO l-1 DO
5)    Set V={ node n in T such that n
       has at least one unused
       adjoining address}
6)    Pick a node n in V in a
       uniformly random manner.
7)    Randomly pick a NULL adjoining
       address a in elementary tree n.
8)    Among all β-trees in the
       auxiliary trees of G_lex
       that can adjoin to a,
       choose a tree t.
9)    Adjoin t to a in T and update T.
10)   ENDFOR
11)  Set individual i-th as T
12)  ENDFOR
```

While the search space for a TAG-based representation is a set of TAG derivation trees, fitness evaluation is carried out on the derived trees decoded from them. Thus the TAG-based representation provides a natural genotype-phenotype mapping, in which genotypes are TAG derivation trees, and phenotypes are their corresponding derived trees. A TAG-based grammar guided genetic programming (called TAG3P) using sub-tree mutation and crossover was described in detail in [26]. For the present purposes, we require only the initialisation procedure.

2) Initialisation Procedure: We define here the algorithm for creating an initial random population of individual (G_{lex} derivation trees). It is an iteration of the process for generating an individual (G_{lex} derivation tree) at random. The process starts with choosing a random size in a predefined range of integer numbers. Then, it proceeds by first randomly picking an α-tree from the initial tree set in G_{lex} to make an initial G_{lex} derivation tree. This derivation tree is subsequently adjoined with β-trees drawn at random from the auxiliary tree set in G_{lex} using adjunction at random places. This process finishes when the randomly chosen size is reached. The initialization procedure for TAG3P is given in table I

where MINSIZE and MAXSIZE are adjustable parameters for designating the range of individual sizes. For this algorithm to be useful, we need a guarantee that the process always stops with the desired result. However, it may not be obvious that this process will finish and give a population of valid G_{lex} derivation trees. We present a proof that this is the case.

Theorem 1: Assuming that every α-tree and every β-tree in G_{lex} can be adjoined by at least one β-tree, the initialisation procedure above always finishes and gives the desired results. *Proof:* The only steps in the algorithm which could fail are steps 5 to 8 (step 3 succeeds because the definition of TAG requires the set of α-trees in G_{lex} to be non-empty, while the

other steps are simply housekeeping).

In step 5, if $j = 1$, then V=$\{\alpha_1\}$ (since by hypothesis, α_1 has an open adjoining address). Otherwise, the last-added β-tree in T must be in V (again by hypothesis). In either case, V is non-empty, so that steps 6 and 7 will also succeed; step 8 then succeeds on the basis of the previous conclusions.

The set V is at most linear in the size of the tree, and the selection process is repeated I-1 times, so that the time complexity for randomly creating an individual from (G_{lex}) is quadratic in its size:

Corollary of proof: The time complexity for initialising an individual is $O(l^2)$, where l is the size of the individual. Therefore, the time complexity of the initialising procedure is M $\times l^2$, where M is the population size.

3) TAG Operators: The non-fixed-arity property permits the definition of a wide range of operators. It is straightforward to define analogues of the wide range of sub-tree crossover and mutation operators used in standard tree-based GP [26]. Equally important, it is simple to define more biologically-motivated operators such as translocation and replication [27]. In the current context, we argue that the cause of GP's structural difficulty problem is the highly discontinuous nature of the two main operators, sub-tree crossover and mutation. To investigate this, we make use of TAG's flexibility to introduce point insertion and deletion operators, which by their local nature permit more fine-grained search of the structure space, than is possible with the standard GP operators.

- Point Insertion
  If the size of the considered individual is less than MAXSIZE, insertion randomly selects a node on the frontier of the derivation tree, and adjoins a new β-tree to it
- Point Deletion
  If the size of the individual is greater than 1, deletion selects a β-tree on the frontier of the derivation tree, and deletes it.

4) Properties of the TAG Genotype-Phenotype Mapping: It is clear that point insertion and deletion are local operators in the genotype (derivation tree) space - using edit distance as a metric [25], [28], the distance between parent and child is one. However TAG representation relies on a genotype to phenotype mapping. We would like to know that point insertion and deletion act as local operators in the phenotype space. This requires us to demonstrate that the mapping satisfies Palmer's [29] locality property, namely that small changes in genotype result in small changes in phenotype. After "redundancy" [32], [41], [43], [44], the "locality property" is possibly the most studied aspect of genotype-phenotype mapping in the field of evolutionary algorithms (EAs). A large number of EA works have shown the importance, across many contexts, of the locality property for mappings from genotype to phenotype spaces. [11]–[13], [24], [30]–[32], [40], [46]. In this paper, the locality property guarantees that, if we can design local structure-editing operators on the genotype space, that the effect of the corresponding structure change in the phenotype space is also local.

A first step is to show that the mapping from derivation tree to derived tree satisfies the locality property. Recall that the tree edit distance between two labeled trees is defined as the length of the shortest sequence of editing operations that transforms one tree to the other. The editing operations are deleting a node, inserting a node, or changing the label of a node. Each basic operation has the same cost, assigned as 1 unit. With the editing distance used as the tree metric, the mapping between TAG-derivation trees and derived trees is Lipschitzian [33], so that the mapping has the locality property.

Theorem 2: For every TAG G_{lex}, suppose that f is the map used to decode G_{lex} derivation trees into the corresponding derived trees. Then there is a fixed constant $M > 0$ such that, for all pairs of G_{lex} derivation trees u, v,

$$d(f(u), f(v)) \leq M \times d(u, v) \tag{4}$$

where $f(u)$ and $f(v)$ are the two derived trees corresponding to u and v respectively, and d is the tree edit distance.

Proof: Let M be the maximal number of nodes in any of the elementary trees E of G_lex. If the tree edit distance between u and v is k, then there are k operations involving the addition and/or deletion of nodes required to transform u to v (and vice versa). Each node in u (or v) is labeled by an elementary tree $t \in E$, and t contains at most M nodes. Moreover, since an adjunction between two elementary trees does not change the meaning or the number of nodes, it follows that the number of node differences (using node addition, deletion, or relabelling) between $f(u)$ and $f(v)$, is at most $M \times k$. Thus the tree editing distance between $f(u)$ and $f(v)$ is less than or equal to $M \times d(u, v)$.

5) TAG Representation for the LID Problem: Daida's LID problem was described in Section II. Here, we describe the specific implementation used to encode it into a TAG representation. The context-free grammar is as follows:
$G = \{N = \{S\}, T = \{join, leaf\}, P, \{S\}\}$ where the rule set P is defined as:
$S \rightarrow SjoinS$
$S \rightarrow leaf$

The corresponding LTAG (using Joshi's algorithm - [19]) is $G_{lex} = \{V = \{S\}, T = \{join, leaf\}, I, A)$ where $I \cup A$ is shown in Figure 9.

With this grammar, the mapping from the derived tree to the corresponding GP tree representation (i.e. the phenotype) is also Lipschitzian: the derived tree has a skeleton of non-terminals S, to each of which is attached exactly one terminal (either join or leaf). If each terminal is moved upwards to replace its corresponding non-terminal S, we obtain the standard GP structure tree. Thus, the edit distance between two derived trees is exactly twice the distance between the corresponding GP structure trees. Since the composition of two Lipschitzian mappings is also Lipschitzian, it follows that, for this problem using TAG representation, that the overall genotype-phenotype mapping is Lipschitzian.

In Figures 9, the β-trees for the LID problem have three adjoining addresses, namely at the root node, at the foot, and at the lower node (labeled with S). In the TAG literature, adjunction at the root and foot nodes in a β-tree are usually

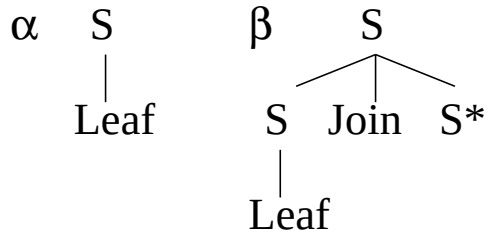

Fig. 9. Elementary trees of G_{lex} for the LID problem

not used simultaneously, since it creates some redundancy in the mapping between TAG-derivation trees and TAG-derived trees. To investigate the effect of this redundancy of the genotype-to-phenotype mapping on the LID problem, we divided the TAG experiments into two sets of runs. In the first set, we used only two adjoining addresses (excluding the root node: 2-ADD), while in the second, all three adjoining addresses were used (3-ADD).

B. Search Algorithm

For standard tree-based GP, the LID problem is an extremely difficult search problem. We have argued that this is a result of the operators available, and that the addition of point insertion and deletion operators should greatly ameliorate this difficulty. To demonstrate just how effective the operators are in smoothing the fitness landscape, we use a naive stochastic hill-climbing search (TAG-HILL), which would be readily caught in any local optima remaining in the fitness landscape. Specifically, the algorithm is a (1+1) evolutionary algorithm, i.e. the algorithm uses a population of 1. At each generation, either point insertion or point deletion is selected (with a probability of 0.5). An individual is then generated by a random application of the selected operator and then, if it has better fitness than its parent, it replaces the parent, otherwise it is discarded. This loop is repeated for the specified number of steps.

IV. Experiments

In our experiments, the maximal number of steps in TAG-HILL is set to 100000 for each run. This gives the same total number of evaluations as in [10], 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 [10]. 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_{target} was varied from 16 to 32768. For each t_{target} in [16..1024], we carried out 100 runs, while for [1025..32768], we ran 100 trials for each point of the 118 equi-sampled points in that interval. Although the settings of W_{depth} and $W_{terminal}$ do not affect TAG-HILL search, we set them the same as in [10], i.e. as 30 and 70 respectively.

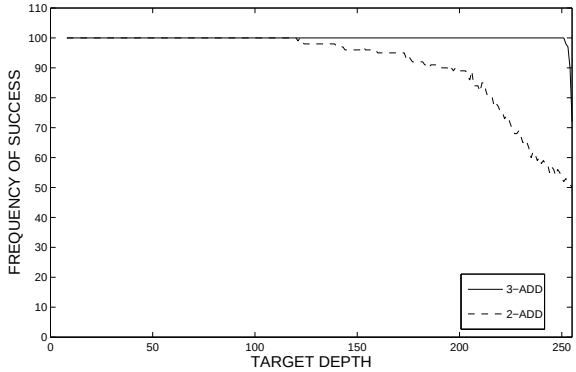

Fig. 10. Results of TAG-HILL on the 'horizontal cut'

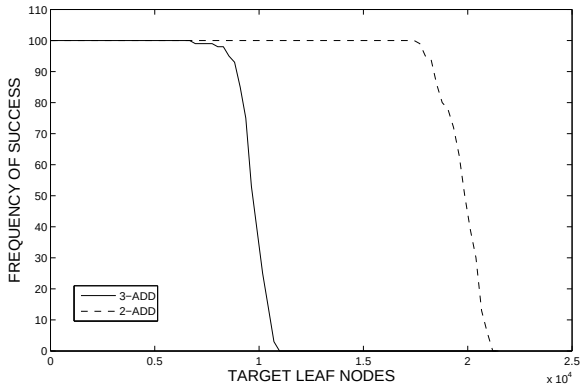

Fig. 11. Results of TAG-HILL on the 'vertical cut'

V. Results and Discussion

Figures 10 and 11 show the proportion of successful runs for TAGHILL (2-ADD and 3-ADD) based on the 275200 runs conducted (137600 each for 2-ADD and 3-ADD).

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 3-ADD solved all the points with 100% reliability, with the exception of the four rightmost points (where the frequencies of success were 98%, 92%, 80%, and 72% respectively). TAG-HILL 2-ADD found the solution with 100% reliability for d_{target} up to 120, and solved the other problem instances with proportions of success ranging from 50% to 98%. By comparison, as shown in Figure 3, GP could only reliably find solutions within the range $12 < d_{target} < 70$. For the ranges $9 < d_{target} < 12$ and $70 < d_{target} < 100$ GP could sometimes find solutions, but for the ranges d_{target} = 8 or 9 and $d_{target} > 100$, GP failed to find any solutions.

For the results on the vertical cut family shown in Figure 4, GP runs were unreliable in finding solutions for $t_{target} > 500$, and failed to find any solutions for $t_{target} > 1024$. By contrast, TAG-HILL 3-ADD could solve the problems with 100% reliability for t_{target} up to nearly 8000, and only failed to find solutions when $t_{target} > 11000$. TAG-HILL 2-ADD was even more successful, managing to solve problems with

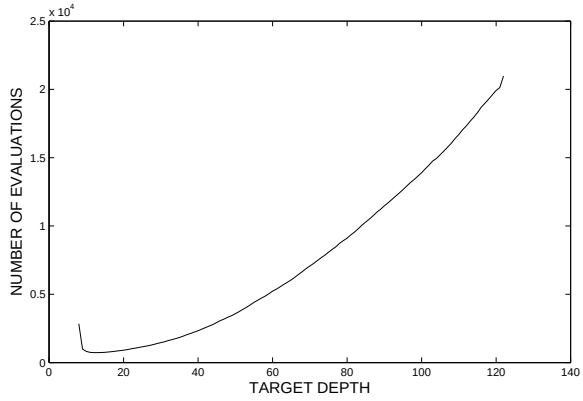

Fig. 12. Average number of fitness evaluations for the "horizontal cut", 2-ADD

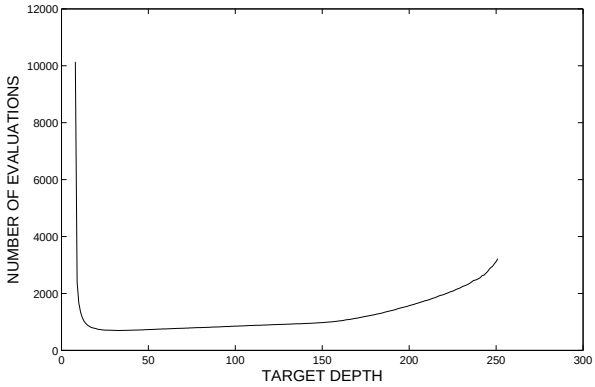

Fig. 13. Average number of fitness evaluations for the "horizontal cut", 3-ADD

100% reliability for t_{target} up to 17700 and only failing to find solutions when $t_{target} > 21000$.

Figures 12, 13, 14 and 15 show the average number of search steps for TAG-HILL (2-ADD and 3-ADD) to find solutions (for those problem instances where 100% success was achieved). 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 structures become exponentially sparse [39]. 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} and d_{target}, out of $\frac{4^{255}}{\sqrt{\pi 255^3}} \approx 2^{497}$ trees in the search space [39].

The results also show that the redundancy in the genotype-to-phenotype mapping helped TAG-HILL 3-ADD perform much better than TAG-HILL 2-ADD on problems of the "horizontal cut" family, while performing much worse on problems of the "vertical cut" family. We explain this as follows. In 3-ADD, there are more available adjoining addresses than in 2-ADD. When an insertion is to occur, in 2-ADD, many of the adjunction addresses higher in the tree will already have been used, so that the probability of selecting a lower adjunction address (and hence deepening the tree) is increased. In 3-ADD, since more addresses higher in the tree are available, the probability of selecting them is reduced more slowly, with fuller trees being the result.

The structural difficulty problem may play a very important part in the understanding of the behaviour of genetic programming. One of the potential applications of this problem lies in explaining the code bloat effect. Code bloat in GP is a well-documented phenomenon, in which the code size of evolved programs increases rapidly during the evolutionary process [1], [2], [23], [45]. One potential cause of bloat may arise from structural difficulty - if the smallest target solutions for a problem lie in regions II and III, GP will probably be unable to find them. However, inserting redundant code into a solution, can move it from regions II and III into region I. Hence, GP search may need to accumulate redundant code to allow it to

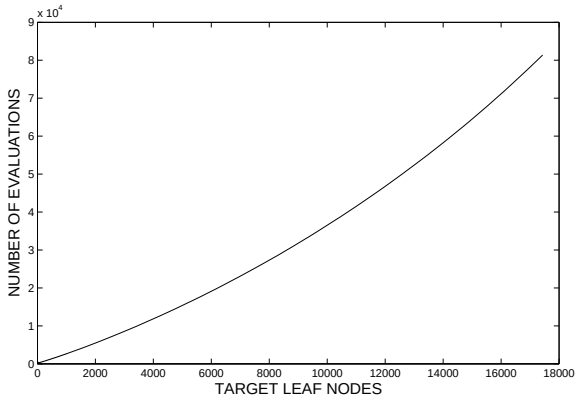

Fig. 14. Average number of fitness evaluations for the "vertical cut", 2-ADD

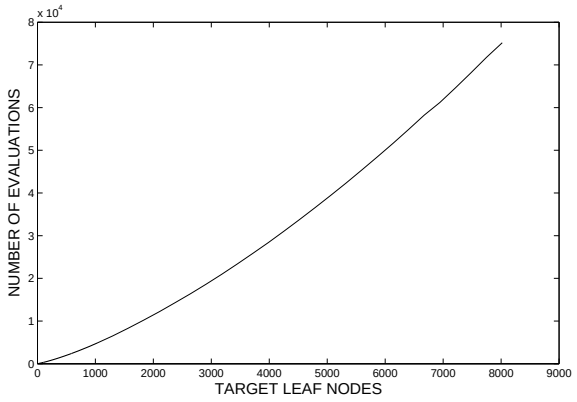

Fig. 15. Average number of fitness evaluations for the "vertical cut", 3-ADD

stay in region I while traversing the fitness landscape towards accessible solutions.

By contrast with the work of Daida and his colleagues, the experimental results here suggest that structural difficulty might arise from the lack of local structure-editing operators in GP. However, there is an alternative potential explanation, namely that it is the hillclimbing search itself which allows our system to solve these problems well. To investigate this further, we applied hill-climbing search to the standard GP representation, using standard subtree mutation as the local search operator. We ran the experiments on the "ends" of the "horizontal cut" and "vertical cut" experiments, the regions where GP failed to find solutions when using genetic search as in [10]. These runs still failed to find any solutions, indicating that the improved results above are not exclusively the result of the hillclimbing search, but instead directly depend on the representation and operators used. It is difficult to find any other explanation of this softening of the structural difficulty problem, than that the local structure-editing operators are the source of the improved performance.

VI. Conclusions and Further Work

We have argued that GP's problem of structural difficulty results from the lack of local structure-editing operators, and have pointed to GP's fixed-arity expression tree representation as the underlying cause. Using a TAG-based representation, we have removed this fixed-arity limitation. In this representation, we were able to design two local structure-editing operators, namely point insertion and deletion. In passing, we note that these operators are not problem-specific, but could be applied to any TAG-representable problem. Applying these operators to Daida's LID problem, we demonstrated that the operators significantly soften the structural difficulty problem in GP. The results also showed that redundancy in the genotype-to-phenotype mapping — created by using redundant adjoining addresses — can affect the efficiency of the operators in finding solutions in the tree structure space.

In this paper, we used the rather naive hill-climbing search strategy. Further work will investigate the behaviour of other adaptive search strategies. On a theoretical level, we are endeavouring to derive a formula to predict the convergence time for TAG-HILL on the LID problem. We are also attempting to analyze the behavior of GP using sub-tree crossover and/or sub-tree mutation on the LID problem, to provide an analytical measure of the structural difficulty problem, and to validate our hypothesis that the difficulties are caused by the structural discontinuity of the standard sub-tree operators in GP.

VII. Acknowledgment

The authors would like to thank Jason Daida for permission to reproduce figures from his papers in this article.

References

[1] W. Banzhaf et al, *Genetic Programming: An Introduction*, Morgan Kaufmann, CA, 1998.

[2] T. Blickle and L. Thiele L., Genetic Programming and Redundancy, in J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation*, 33-38, 1994.

[3] O.A. Cha et al, Characterizing a Tunably Difficult Problem in Genetic Programming, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO 2000)*, Morgan Kaufmann, 395-402, 2000.

[4] N.L. Cramer, A Representation for the Adaptive Generation of Sequential Programs, *Proceedings of an International Conference on Genetic Algorithms and the Applications*, 183-187, 1985.

[5] J.M. Daida et al, Challenges with Verification, Repeatability, and Meaningful Comparisons in Genetic Programming, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Morgan Kaufmann, 64-69, 1997.

[6] J.M. Daida et al, Analysis of Single-Node (Building) Blocks in Genetic Programming, in L. Spector, W.B. Langdon et al, editors, *Advances in Genetic Programming III*, The MIT Press, 217-241, 1999.

[7] J.M. Daida et al, What Makes a Problem GP-Hard? Analysis of a Tunably Difficult Problem in Genetic Programming, *Genetic Programming and Evolvable Machines*, **2**, 165-191, 2001.

[8] J.M. Daida, Limit to Expression in Genetic Programming: Lattice-Aggregate Modeling, *Proceedings of the Congress on Evolutionary Computation (CEC 2002)*, IEEE Press, 273-278, 2002.

[9] J.M. Daida and A.M. Hilss, Identifying Structural Mechanism in Standard GP, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO'2003)*, LNCS 2724, Springer-Verlag, 1639-1651, 2003.

[10] J.M. Daida et al, What Makes a Problem GP-Hard? Validating a Hypothesis of Structural Causes, *Proceedings of Genetic Algorithms and Evolutionary Computation Conference (GECCO 2003)*, LNCS 2724, Springer-Verlag, 1665-1677, 2003.

[11] S. Droste and D. Wiesmann, On Representation and Genetic Operators in Evolutionary Algorithms, Accessed at: citeseer.ist.psu.edu/droste98representation.html on 30 oct 2004.

[12] J. Gottlieb and G. R. Raidl, Characterizing Locality in Decoder-Based EAs for the Multidimensional Knapsack Problem, *Proceedings of Artificial Evolution*, LNCS 1829, Springer-Verlag, 38-52, 1999.

[13] C. Igel, Causality of Hierarchical Variable Length Representations, *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, IEEE Press, 324-329, 1998.

[14] A.K. Joshi et al, Tree Adjunct Grammars, *Journal of Computer and System Sciences*, **10** (1), 136-163, 1975.

[15] A.K. Joshi, Constraints on Structural Descriptions: Local Transformation, *SIAM Journal of Computing*, June, 1977.

[16] A.K. Joshi, How Much Context-sensitivity is Necessary for Characterizing Structural Description, in D. Dowty et al, editors, *Natural Language Processing - Theoretical, Computational and Psychological Perspectives*, Cambridge University Press, 1985.

[17] A.K. Joshi, An Introduction to Tree Adjoining Grammars, in A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[18] A.K. Joshi et al, The Convergence of Mildly Context-Sensitive Grammar Formalisms, in P. Sells et al, editors, *Foundation Issues in Natural Language Processing*, MIT Press, MA, 1991.

[19] A.K. Joshi and Y. Schabes, Tree Adjoining Grammars, in G. Rozenberg and A. Saloma, editors, *Handbook of Formal Languages*, Springer-Verlag, 69-123, 1997.

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

[21] A. Kroch and A.K. Joshi, Linguistic Relevance of Tree Adjoining Grammars, *Technical Report MS-CIS-85-18*, Department of Computer Science and Information Science, University of Pennsylvania, April, 1985.

[22] A. Kroch, Unbounded Dependencies and Subjacency in a Tree Adjoining Grammar, In A. Manaster-Ramer, editor, *Mathematics of Language*, John Benjamins, Amsterdam, 1987.

[23] W.B. Langdon and R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, Germany, 2002.

[24] P.K. Lehre and P.C. Haddow, Developmental Mapping and Phenotypic Complexity, *Proceedings of Congress on Evolutionary Computation (CEC 2003)*, IEEE Press, 62-65, 2003.

[25] S.Y. Lu, The Tree-to-Tree Distance and Its Application in Cluster Analysis, *IEEE Transaction on PAMI*, **1**(2), 219-222, 1979.

[26] Nguyen Xuan Hoai et al, Some Experimental Results with Tree Adjunct Grammar Guided Genetic Programming, *Proceedings of the Fifth European Conference on Genetic Programming*, LNCS 2278, Springer-Verlag, 328-337, 2002.

[27] Nguyen Xuan Hoai et al, Genetic Transposition in Tree Adjoining Grammar-Guided Genetic Programming: The Relocation Operator,

Proceedings of the 5th International Conference on Simulated Evolution and Learning (SEAL 04), 2004.
[28] U.M. O'Reilly, Using a Distance Metric on Genetic Programs to Understand Genetic Operators, *Late Breaking Papers at the 1997 Genetic Programming Conference*, 199-206, 1997.
[29] C.C. Palmer and A. Kershenbaum, Representing Trees in Genetic Algorithms, *Proceedings of the First IEEE Conference on Evolutionary Computation*, 379-384, 1994.
[30] I. Rechenberg, *Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution*, Fromman-Hozlboog Verlag, Stuttgart, 1973.
[31] J.P. Rosca and D.H. Ballard, Causality in Genetic Programming, *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Morgan Kaufmann, 1995.
[32] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, Physica-Verlag, 2002.
[33] W. Rudin, *Principles of Mathematical Analysis*, Third Edition, McGraw-Hill, 1976.
[34] Y. Schabes, *Mathematical and Computational Aspects of Lexicalized Grammars*, PhD Thesis, Department of Computer and Information Science, University of Pennsylvania, 1990.
[35] Y. Schabes, Lexicalized Context-Free Grammars, Technical Report TR93-01, Mitsubishi Electric Research Laboratories, Cambridge Center, 1993.
[36] Y. Schabes and R.C. Waters, Lexicalized Context-Free Grammar: A Cubic-Time Parsable, Lexicalized Normal Form for Context-Free Grammar That Preserves Tree Structure, Technical Report TR93-04, Mitsubishi Electric Research Laboratories, Cambridge Center, 1993.
[37] Y. Schabes and S. Shieber, An Alternative Conception of Tree-Adjoining Derivation, *Computational Linguistics*, **20** (1), 91-124, 1994.
[38] Y. Schabes and R.C. Waters, Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced, *Computational Linguistics*, **20** (1), 479-513, 1995.
[39] R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, 1996.
[40] B. Sendhoff et al, A Condition for the Genotype-Phenotype Mapping: Causality, *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA 97)*, Morgan Kaufmann, 73-80, 1997.
[41] M. Shackleton et al, An Investigation of Redundant Genotype-Phenotype Mappings and Their Role in Evolutionary Search, *Proceedings of the Congress on Evolutionary Computation CEC 2000*, IEEE Press, 493-500, 2000.
[42] V. Shanker, *A Study of Tree Adjoining Grammars*, PhD. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1987.
[43] R. Shipman, Genetic Redundancy: Desirable or Problematic for Evolutionary Search ?, *Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, 1-11, 1999.
[44] R. Shipman, M. Shackleton, I. Harvey, The Use of Neutral Genotype-Phenotype Mappings for Improved Evolutionary Search, *BT Technology Journal*, **18** (4), 103-111, 2000.
[45] T. Soule and J.A. Foster, Effects of Code Growth and Parsimony Pressure on Population in Genetic Programming, *Evolutionary Computation*, **6**(4), 293-309, 1999.
[46] P. Stagge and C. Igel, Structure Optimization and Isomorphisms, *Theoretical Aspects of Evolutionary Computing*, Springer-Verlag, 2000.
[47] D.J. Weir, *Characterizing Mildly Context-Sensitive Grammar Formalisms*, PhD. Thesis, Department of Computer and Information Science, University of Pennsylvania, 1988.
[48] P.A. Whigham, *Grammatical Bias for Evolutionary Learning*, PhD thesis, University of New South Wales, 1996.

Nguyen, Xuan Hoai received his BSc degree in Computer Science from Hanoi University, Vietnam in 1995 and his MSc degree in Mathematics for Computer and Computing Systems from Vietnam National University (VNU) in 1997. He joined the Department of Information Technology at the Viet-Nam Military Technical Academy in 1997 as a lecturer. He has recently completed his PhD degree in Computer Science at the Australian Defence Force Academy (ADFA), University of New South Wales, and returned to the VietNam Military Technical Academy. He is a member of the Complexity, Interaction, and Adaptation (CIA) Group at ADFA, and a student member of the IEEE. His research interests include Genetic Programming, Grammar Guided Evolutionary Learning, Evolutionary Computation, and Fractal Theory.

Bob McKay received his BSc in Pure Mathematics from the Australian National University in 1971, and his PhD in the theory of computation from the University of Bristol, UK, in 1976.

He was a Research Scientist in computer typesetting at the (Australian) Commonwealth Scientific and Industrial Research Organisation from 1976 to 1985, when he joined the University of New South Wales at the Australian Defence Force Academy as a Lecturer and subsequently Senior Lecturer. He moved to Seoul National University, Korea, as an Associate Professor in 2005. His research interests lie in artificial intelligence, evolutionary computation and ecological modelling.

He is an Associate Editor of the IEEE Transactions on Evolutionary Computation, an editorial board member of Genetic Programming and Evolvable Machines, and an advisory board member of the International Journal of Knowledge-Based and Intelligent Engineering Systems

Daryl Essam received his PhD in fractal image processing from the University of New South Wales at the Australian Defence Force Academy in ?2000?.

His research focuses on new algorithms for genetic programming. In particular the sub-fields of grammars, diversity, probabilistic approaches and multi-objective optimisation. He is currently employed as a lecturer at the Australian Defense Force Academy, a campus of the University of New South Wales, Australia.