Solving the Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Comparative Results

N.X.Hoai[1], R.I. McKay[2], and D. Essam[2]
School of Computer Science,
University of New South Wales,
ADFA campus, Canberra,
ACT 2600, Australia,
[1] x.nguyen@student.adfa.edu.au
[2] rim, daryl@cs.adfa.edu.au

Abstract - In this paper, we show some experimental results of tree-adjunct grammar guided genetic programming [6] (TAG3P) on the symbolic regression problem, a benchmark problem in genetic programming. We compare the results with genetic programming [9] (GP) and grammar guided genetic programming [14] (GGGP). The results show that TAG3P significantly outperforms GP and GGGP on the target functions attempted in terms of probability of success. Moreover, TAG3P still performed well when the structural complexity of the target function was scaled up.

I. INTRODUCTION

Tree adjunct grammar guided genetic programming [6] (TAG3P) is a grammar guided genetic programming system that uses tree adjunct grammars along with context free grammars as means to set bias for the evolutionary process. The preliminary results in [4], [5] indicated that TAG3P works well on the symbolic regression problem. In this paper we experiment with GP, GGGP and TAG3P on the symbolic regression problem with different target functions to compare their performances and to observe how well they solve the problem when the structural complexity of the target function is scaled up. The organization of the remainder of the paper is as follow. In section 2, we give some basic concepts of GP, GGGP, and TAG3P. Section 3 describes the symbolic regression problem. Section 4 contains our experimental setup. The results will be given and discussed in section 5. Section 6 concludes the paper and discusses some future work.

II. BACKGROUNDS

In this section, we briefly overview some basic components and operations of the three different genetic programming systems, namely, canonical genetic programming [9] (GP), grammar guided genetic programming [14] (GGGP) and tree adjunct grammar guided genetic programming [6] (TAG3P).

II.1 *Genetic Programming*

Genetic programming (GP) can be classified as an evolutionary algorithm, in which computer programs are the evolutionary targets. An early definition, model, techniques and problems of genetic programming can be found in [9]. For a good survey of genetic programming, [1] is recommended. A basic genetic programming system consists of five basic components [9]: representation for programs (called genome structure), a procedure to initialize a population of programs, a fitness to evaluate the performance of the program, genetic operators, and parameters. In [9], the structure of programs is the structured tree of S-expressions; fitness of a program is evaluated by its performance; main genetic operators are reproduction, crossover, and mutation. Reproduction means some programs are copied to the next generation based on their fitness, crossover can be carried out between two tree-based programs by swapping two of their sub-trees,[1] and a tree-based program can be mutated by replacing one of its sub-trees by a randomly generated tree. Parameters are population size, maximum number of generations and probabilities for genetic operators. The evolutionary process is as follows. At the beginning, a population of tree-based programs is randomly generated. Then, the new population is created by applying genetic operators to the individuals chosen from the existing population based on their fitness. This process is repeated until the desired criteria are satisfied or the number of generations exceeds the maximum number of generation. GP has been used successfully in generating computer programs for solving a number of problems in a wide range of areas [1].

II.2 Grammar Guided Genetic Programming

Grammar guided genetic programming systems are genetic programming systems that use grammars to set syntactical constraints on programs. The use of grammars also helps these genetic programming systems to overcome the closure requirement in canonical genetic programming, which cannot always be fulfilled [14].

Using grammars to set syntactical constraints was first introduced by Whigham [14] where context-free grammars were used. We shall refer Whigham's system as GGGP for the rest of the paper. Basically, GGGP has the same components and operations as in GP; however, there are a number of significant differences between the two systems. In GGGP, a program is represented as its derivation tree in the context free grammar. Crossover between two programs can only be carried out by swapping their two sub-derivation trees that start with the inner nodes labelled by the same non-terminal symbol in the

[1] The ideas of using tree-based representation of chromosomes and swapping sub-trees as crossover operator was first introduced in [2].

grammar. In mutation, a sub-derivation tree is replaced by a randomly generated sub-derivation tree that is derived from the same non-terminal symbol. GGGP demonstrated positive results on the 6-multiplexer problem and subsequently on a wide range of other problems.

II. 3 *Tree Adjunct Grammar Guided Genetic Programming*

Tree adjunct grammar guided genetic programming [6] (TAG3P) uses tree adjunct grammars along with context free grammars to set syntactical constraints as well as search bias for the evolution of programs. In this subsection we will first give the basic concepts of tree adjunct grammars then the basic components of TAG3P.

II.3.1 *Tree Adjunct Grammars*

Tree-adjunct grammars are tree-rewriting systems, defined in [7] as follows:

Definition 1: a tree-adjunct grammar comprises of 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. I is a set of trees called initial trees. An initial tree is defined as follows: the root node is S; all interior nodes are labelled by non-terminal symbols; each node on the frontier is labelled by a terminal symbol. A is a finite set of trees called auxiliary trees, which can be defined as follows: internal nodes are labelled by non-terminal symbols; a node on the frontier is labelled by a terminal or non-terminal symbol; there is a special non-terminal node on the frontier called the foot node. The foot node must be labelled by the same (non-terminal) symbol as the root node of the tree. We will follow the convention in [8] to mark the foot node with an asterisk (*).

The trees in $E= I \cup A$ are called elementary trees. Initial trees and auxiliary trees are denoted α and β respectively; and a node labelled by a non-terminal (resp. terminal) symbol is sometime called a non-terminal (resp. terminal) node. An elementary tree is called X-type if its root is labelled by the non-terminal symbol X.

The key operation used with tree-adjunct grammars is the adjunction of trees. Adjunction can build a new (derived) tree γ from an auxiliary tree β and a tree α (initial, auxiliary or derived). If a tree α has a non-terminal node labelled A, and β is an A-type tree then 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 this 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 1.

The tree set of a TAG can be defined as follows [7]:

T_G = {all tree t / t is completed and t is derived from some initial trees}

A tree t is completed, if t is an initial tree or all of the leaf nodes of t are non-terminal nodes; and a tree t is said to be derived from a TAG G if and only if t results from an adjunction sequence (the derivation sequence) of the form: $\alpha\ \beta_1(a_1)\ \beta_2(a_2)...\ \beta_n(a_n)$, where n is an arbitrary integer, α , β_i (i=1,2..n) are initial and auxiliary trees of G and a_i (i=1,2..n) are node address where adjunctions take place. An adjunction sequence may be denoted as (*). The language L_G generated by a TAG is then defined as the set of yields of all trees in T_G.

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

The set of languages generated by TAGs (called TAL) is a superset of context-free languages; and is properly included in indexed languages [8]. More properties of TAL can be found in [8]. One special class of tree-adjunct grammars (TAGs) is lexicalized tree-adjunct grammars (LTAG) where each elementary tree of a LTAG must have at least one terminal node. It has been proved that for any context-free grammar G, there exists a LTAG G_{lex} that generates the same language and tree set with G (G_{lex} is then said to strongly lexicalize G) [8].

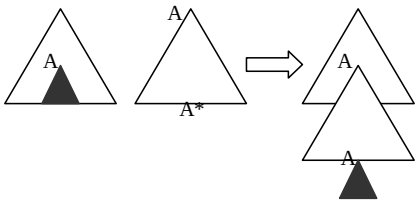

Figure 1. Adjunction.

II.3.2 *Tree Adjunct Grammar Guided Genetic Programming*

In [6], we proposed a grammar guided genetic programming system called TAG3P, which uses a pairs consisting of a context-free grammar G and its corresponding LTAG G_{lex} to guide the evolutionary process. The main idea of TAG3P is to evolve the derivation sequence in G_{lex} (genotype) rather than evolve the derivation tree in G as in [14]. Therefore, it creates a genotype-to-phenotype map. As in GP [9], TAG3P comprises of the following five main components:

Program representation: a modified version of the linear derivation sequence (*), but the adjoining address of the tree β_i is in the tree β_{i-1}. Thus, the genome structure in TAG3P is linear and length-variant. Although the language and the tree set generated by LTAGs with the modified derivation sequence is yet to be determined, we have found pairs of G and G_{lex} conforming to that derivation form for a number of standard problems in genetic programming [4], [5].

Initialization procedure: a procedure for initializing a population is given in [6]. To initialize an individual, TAG3P starts with selecting a length at random; next, it picks up randomly an α tree of G_{lex} then a random sequence of β trees and adjoining addresses. It has been proved that this procedure can always generate legal genomes of arbitrary and finite lengths [6].

Fitness Evaluation: the same as in canonical genetic programming [9].

Genetic operators: in [6], we proposed two types of crossover operators, namely one-point and two-point crossover,

and three mutation operators, which are replacement, insertion and deletion. The crossover operators in TAG3P are similar to those in genetic algorithms; however, the crossover point(s) is chosen carefully so that only legal genomes are produced. In replacement, a gene is picked up at random and the adjoining address of that gene is replaced by another adjoining address (adjoining address replacement); or, the gene itself is replaced by a compatible gene (gene replacement) so that the resultant genome is still valid. In insertion and deletion, a gene is inserted into or deleted from the genome respectively. With these carefully designed operators, TAG3P is guaranteed to produce only legal genomes. Selection in TAG3P is similar to canonical genetic programming and other grammar-guided genetic programming systems. Currently, reproduction is not employed by TAG3P.

Parameters: minimum length of genomes, MIN_LENGTH, maximum length of genomes MAX_LENGTH, size of population - POP_SIZE, maximum number of generations – MAX_GEN and probabilities for genetic operators.

Some analysis of the advantages of TAG3P can be found in [4]-[6].

II.4 *Other Grammar Guided Genetic Programming Systems*

Wong and Leung [15] used logic grammars to combine inductive logic programming and genetic programming. They have succeeded in incorporating domain knowledge into logic grammars to guide the evolutionary process of logic programs.

Ryan and his co-workers [13] proposed a system called grammatical evolution (GE), which can evolve programs in any language, provided that this language can be described by a context-free grammar. Their system differs from Whigham's system in that it does not evolve derivation trees directly. Instead, genomes in GE are binary strings representing eight-bit numbers; each number is used to make the choice of the production rule for the non-terminal symbol being processed. GE has been shown to outperform canonical GP on a number of problems [10].

III. SYMBOLIC REGRESSION PROBLEM

The symbolic regression problem can be stated as finding a function in symbolic form that fits a given finite sample of data [9]. In [9], the problem is restricted to finding a function of one independent variable. As in [9], the function set for GP in this paper is {+, -, *, /, sin, cos, Log, Exp}, and the terminal set is {X}. The problem space for GGGP and TAG3P can be described by a finitely ambiguous context-free grammar G and the corresponding lexicalized tree-adjunct grammar G_{lex} as follows [4].

The context-free grammar for the symbolic regression problem: G = (N={EXP, PRE, OP, VAR,},T= {X, sin, cos, log, ep, +, -, *, /, (,)},P,{EXP}} where ep is the exponential function, and the rule set P={EXP→EXP OP EXP, EXP→PRE (EXP), EXP→VAR, OP→+, OP→ - , OP→*, OP→/, PRE→ in, PRE→cos, PRE→log, PRE→ep, VAR→ X}.

The tree adjunct grammar for the symbolic regression problem: G_{lex}= {N={EXP, PRE, OP,VAR},T={X, sin, cos, log, ep,+, -, *, /, (,)}, I, A) where I∪ A is as in Figure 2.

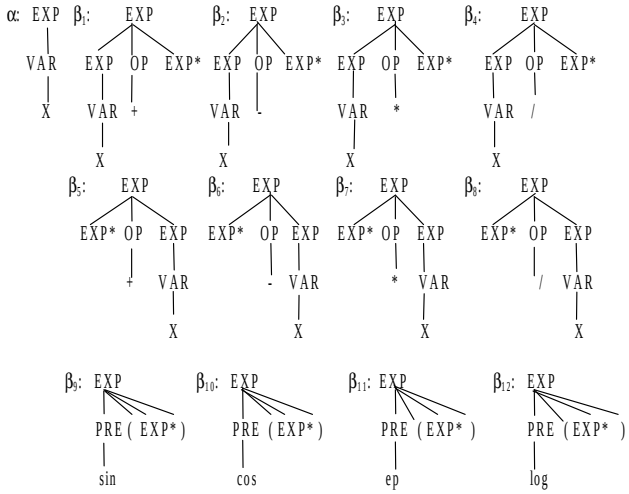

Figure 2 Elementary trees for G_{lex}.

A variation of the simple symbolic regression above is known as the trigonometric identities problem [9], where the basic trigonometric function of the target function does not appear in the function set. We will follow [9] in setting the function set for GP is {+, -, *, /, sin}, and the terminal set is {X, 1.0}. The target function is cos(2X) as in [9]. The context free grammar G and its corresponding LTAG G_{lex} for GGGP and TAG3P are as follows [5].

The context-free grammar for the problem of finding trigonometric identities: G = (N, T, P, {EXP}). Where N={EXP, PRE, OP, VAR, NUM} is the set of non-terminal symbols; T= {X, sin, +, -, *, /, (,), 1.0}, P, {EXP}) is the set of terminal symbols; EXP is the start symbol; and the rule set P={EXP→EXP OP EXP, EXP→ PRE(EXP), EXP→VAR, EXP→ NUM, OP→ +, OP→ - , OP→ *, OP→ /, PRE→ sin , VAR→ X, NUM→ 1.0}.

The tree-adjunct grammar for the problem of finding trigonometric identities G_{lex}= (N, T, I, A). Where N={EXP, PRE, OP, VAR, NUM} is the set of non-terminal symbols; T={ X, sin, +, -, *, /, (,), 1.0} is the set of terminal symbols; and I and A are the sets of initial and auxiliary trees respectively. The set of the elementary trees I ∪ A is as in Figure 3.

IV. EXPERIMENTAL DESIGN

Five experiments were conducted. In the first four experiments, we tried all three systems on the symbolic regression problem with four different target functions namely, $F_1=X^2+X$, $F_2=X^3+X^2+X$, $F_3=X^4+X^3+X^2+X$, and $F_4=X^5+X^4+X^3+X^2+X$.

The aim of the experiments was not only to compare the abilities of GP, GGGP and TAG3P in inducing the target

function from the sampled data but also to observe their efficiency when the structural complexity of the target function was increased. It should be noted that each F_i above (i=2,3,4) can be represented as F_{i-1}*X+X.

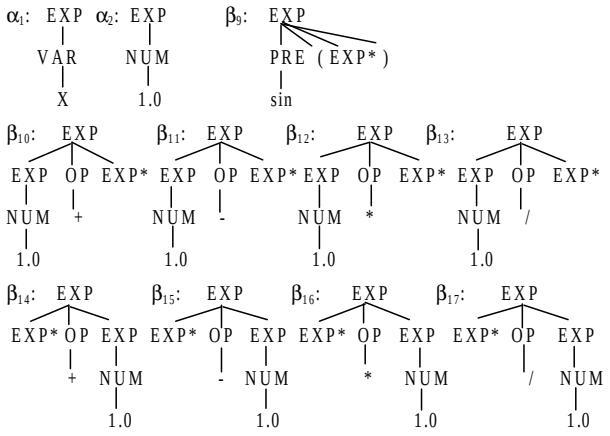

Figure3. Elementary trees for G_{lex} the β_1-β_8 are the same as the β_1-β_8 in Figure 2.

In the fifth experiment, all three systems were experimented on the trigonometric identities problem with cos(2X) as the target function. The aim was to derive the three alternative identities, namely, $1\text{-}2\sin^2(X)$, $\cos(2X+\pi/2)$ and $\cos(\pi/2\text{-}2X)$. Table 1 summarises the experimental design for the three systems.

V. RESULTS

In each experiment, 150 runs were conducted; 50 runs for each system. During these runs GP, GGGP and TAG3P used the same data set. The number of successful runs for each system is summarised in table 2.

In the first experiment, the three system discovered the exact solution almost all the time. One of the reasons for this success is that the structural complexity of the target function is small. Figure 4 depicts their cumulative frequencies. GP and TAG3P found the solution at quite early generations whereas GGGP found the solution somewhat later.

The cumulative frequencies in the second experiment were shown in Figure 5. The efficiency of TAG3P was maintained when the target function was changed from F_1 to F_2. In contrast, the probability of success for GP and GGGP decreased dramatically. In this experiment, GGGP slightly outperformed GP.

Figure 6 shows the cumulative frequencies of all three systems for target function F3 in the third experiment.

Once again, there was a big fall in the probability of success of GP and GGGP where the target function was made slightly more complex in structure. On the contrary, TAG3P still handled the increase in the structural complexity of the target function.

The cumulative frequencies for function F4 are recorded in Figure 7. The results confirm those for F3: TAG3P significantly outperformed GP and GGGP and still worked well when the target function was once again made more complex in terms of structural complexity.

In the last experiment, we tried all three systems on the trigonometric identities problem. The target function was cos (2X). This target function is harder to induce than in the last four experiments because the function cos was excluded from the function set in this experiment. Consequently, the number of successful runs for GP, GGGP and TAG3P lower. Figure 8 depicts the cumulative frequencies of the three systems.

Objective	Find a function of one independent variable and one dependant variable that fits a given sample of 20 (x_i, y_i) data points, where the target functions are F_1-F_4, cos(2X).
Terminal Operands	X (the independent variable) in experiments 1 to 4; {X, 1.0} in experiment 5.
Terminal Operators	The binary operators are +,-,*,/. The unary operators are sin, cos, exp and log in experiment 1 to 4; and it is only sin in experiment 5.
Fitness Cases	The sample of 20 points in the interval [-1..+1] in experiments 1 to 4, and in the interval $[0..2\pi]$ in experiment 5.
Raw fitness	The sum, taken over 20 fitness cases, of the errors.
Standardized Fitness	Same as raw fitness.
Hits	The number of fitness cases for which the error less than 0.01.
Genetic Operators	Tournament selection, one-point crossover and gene replacement for TAG3P. Tournament selection, normal crossovers and mutations for GP and GGGP.
Parameters	The crossover probability for GP, GGGP, TAG3P is 0.9. The mutation probability for GP and GGGP is 0.1. Replacement probability for TAG3P is 0.01 for experiments 1 to 4 and 0.05 for experiment 5. Tournament size is 3. MAX_GEN is 30 in experiments 1 to 4 and 200 in experiment 5. POP_SIZE=500.
Success predicate	An individual scores 20 hits.

Table1. Experimental setups for GP, GGGP and TAG3P.

Target functions	GP	GGGP	TAG3P
F1	47(94%)	46(92%)	50(100%)
F2	30(60%)	32(64%)	50(100%)
F3	21(42%)	24(48%)	48(96%)
F4	9(18%)	14(28%)	42(84%)
Cos(2x)	0(0%)	10(20%)	18(36%)

Table2. Number of successful runs for three systems in five experiments.

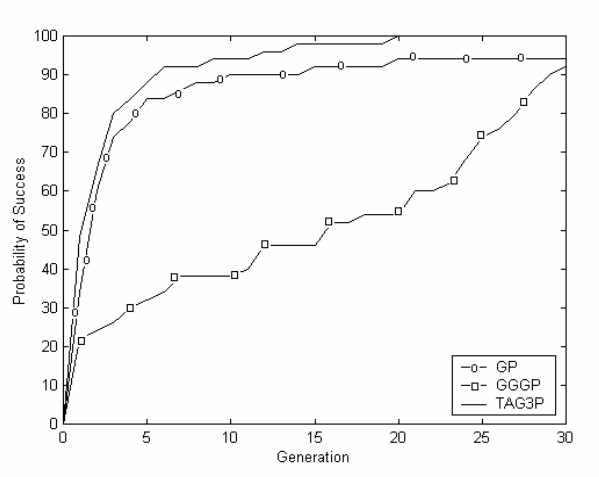

Figure 4. Cumulative frequencies of GP, GGGP and TAG3P in experiment 1, where the target function is X^2+X.

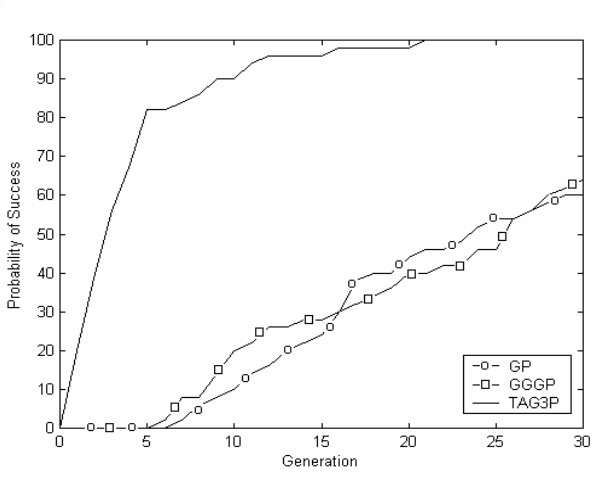

Figure 5. Cumulative frequencies of GP, GGGP and TAG3P in experiment 2, where the target function is X^3+X^2+X.

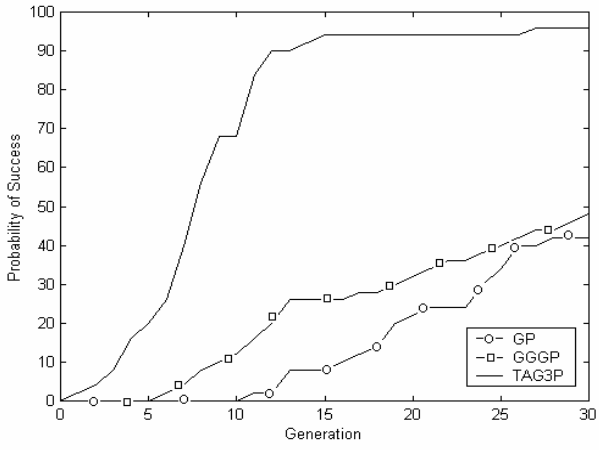

Figure 6. Cumulative frequencies of GP, GGGP and TAG3P in experiment 3, where the target function is $X^4+X^3+X^2+X$.

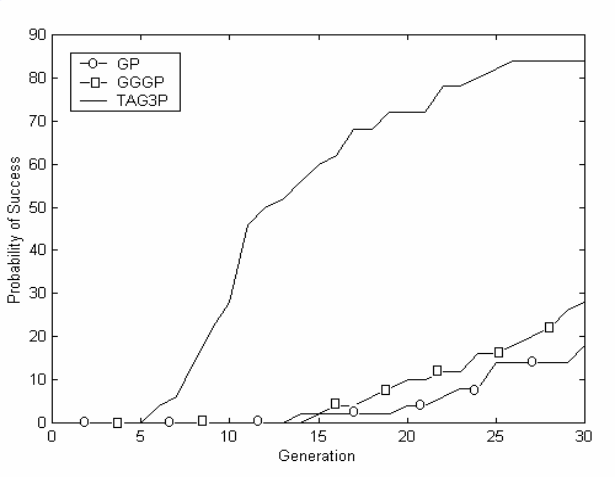

Figure 7. Cumulative frequencies of GP, GGGP and TAG3P in experiment 4, where the target function is $X^5+X^4+X^3+X^2+X$.

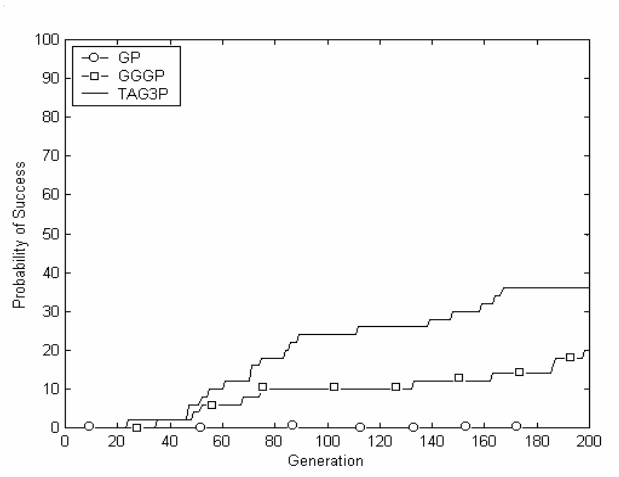

Figure 8. Cumulative frequencies of GP, GGGP and TAG3P in experiment 5, where the target function is cos(2X).

GGGP and TAG3P discovered all the three alternative representations of the cos(2X) function mentioned in the previous section. For GGGP, 1 out of 10 successful runs discovered the exact representation $1-2\sin^2(X)$, and the rest found approximate representations of $\cos(2X+\pi/2)$ and $\cos(\pi/2-2X)$. Here we consider a function F as an approximate representation of the two above functions if it is of the form cos (2X+A) or cos (A-2X) (where A is a constant close to $\pi/2$) it scores all 20 hits, and its raw fitness is smaller than 0.02. Of the 18 successful runs for TAG3P, the number of exact and approximate representations found by were 2 and 16 respectively.

One of the reasons for the good performance of TAG3P on the above problems lies with the superior capability of TAG3P in preserving and combining building blocks [4], [5]. Building blocks [11], [12] are sub-trees, which are not completed. Functionally, they can be viewed as some potential modules.

For example, the atomic building blocks for the target functions in the first four experiments could be X+t and X*t, where t is a parameter representing the incomplete portion of the tree, which are exactly β_1 and β_3 in Figure 2 [4]. During the evolutionary process, these building blocks are preserved and combined to make even better building blocks. For instance, if β_1, β_3 and β_3 are brought together in a chromosome then the corresponding blocks in the phenotype space will be $X+X^2+t$ after adjunction takes place. The atomic building blocks in TAG3P for the trigonometric identities was found in [5] as sin(2x+1+ t), sin(2x+sin(1/sin(1) + t), sin(2x+1+1/sin(1)+ t), sin(1-2x+ t), sin(sin(1/sin(1))-2x+ t), and sin(1/sin(1)+1-2x + t), which result from short sequences of some beta trees in Figure 3.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have empirically shown that, with all the target functions attempted, TAG3P outperforms GP and GGGP significantly on the symbolic regression and its slightly modified version, namely, trigonometric identities. The results also show that the performance of TAG3P compared to GP and GGGP scales better as the structural complexity of the target function is increased.

In [3], we used TAG3P with tree adjoining address replacement as the mutation operator to solve the symbolic regression problem, where the target function was F_3. The result was not as good as in this paper (probability of success was 64%). This was a result of premature convergence. In this paper, we used a more disruptive replacement operator, gene replacement. Consequently, we increased the diversity of the population and got much better results. In future, we will be investigating further more disruptive operators and mechanisms for diversity maintenance.

The current version of TAG3P uses a restricted (linear) form of the derivation sequence, which is sufficient for many context-free languages, but not all. We are currently developing a TAG-based GP system with the most general form of derivation sequence so as to make TAG3P universal for problems with a context-free search space.

References

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

[2] N. L. Cramer, "A representation for the Adaptive Generation of Simple Sequential Programs', Proceedings of an International Conference on Genetic Algorithms and the Applications, pp. 183 – 187, Lawrence Erlbaum Associates, July 1985.

[3] F. Gruau, "On Using Syntactic Constraints with Genetic Programming", Advances in Genetic Programming, The MIT Press, pp. 377-394, 1996.

[4] N.X. Hoai, "Solving The Symbolic Regression Problem with Tree-Adjunct Grammar Guided Genetic Programming: The Preliminary Results", Proceedings of The 5th Autraliasia-Japan Co-Joint Workshop on Evolutionary Computation, pp. 52-61, 2001.

[5] N.X. Hoai, "Solving Trigonometric Identities with Tree Adjunct Grammar Guided Genetic Programming", To appear in the Proceedings of The First International Workshop on Hybrid Intelligent Systems (HIS'01), Adelaide, Australia, 11-12 Dec 2001.

[6] N.X. Hoai and R.I. McKay, "A Framework for Tree Adjunct Grammar Guided Genetic Programming", Proceedings of the Post-graduate ADFA Conference on Computer Science (PACCS'01), pp. 93-99, 2001.

[7] A.K. Joshi, L.S. Levy, and M. Takahashi, "Tree Adjunct Grammars", *Journal of Computer and System Sciences*, Vol. 10:1, pp. 136-163, 1975.

[8] A.K. Joshi and Y. Schabes, "Tree Adjoining Grammars", Handbook of Formal Languages, Springer-Verlag, pp. 69-123, 1997.

[9] J. Koza, Genetic Programming, The MIT Press, 1992.

[10] M. O'Neill and C. Ryan, "Grammatical Evolution: A Steady State Approach", Proceedings of the Second International Workshop on Frontiers in Evolutionary Algorithms, pp. 419-423, 1998.

[11] R. Poli and N.F. McPhee, "Exact Schema Theory for GP and Variable Length Gas with Homologous Crossover", *GECCO*, San Fransisco, pp. 104-111, 2001.

[12] J.P. Rosca, and D.H. Ballard, "Genetic Programming with Adaptive Representations", Technical Report 489, The University of Rochester, Feb 1994.

[13] C. Ryan, J.J. Collin, M. O'Neill, "Grammatical Evolution: Evolving Programs for an Arbitrary Language", Lecture Note in Computer Science 1391, Proceedings of the First European Workshop on Genetic Programming, Springer-Verlag, pp. 83-95, 1998.

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

[15] M.L. Wong and K.S. Leung, "Evolving Recursive Functions for Even-Parity Problem Using Genetic Programming", Advances in Genetic Programming, The MIT Press, pp. 221-240, 1996.