A New Approach to Evaluate GP Schema in Context

Hammad Majeed
Department of Computer Science and Information Systems
University of Limerick
Limerick, Ireland
hammad.majeed@ul.ie

ABSTRACT

Evaluating GP schema in *context* is considered to be a complex, and, at times impossible, task. The tightly linked nodes of a GP tree is the main reason behind its complexity.

This paper presents a new approach to evaluate GP schema in context. It is simple in its implementation with a potential to address well-known GP problems, such as *identification of significant schema*, *dead code (introns)* and *module acquisition* to name a few.

It is based on the principle that the contribution of a schema can be evaluated by *neutralizing* the effect of the schema in the tree containing it (container-tree) and then checking its effect on the container-tree's fitness. Its usefulness is empirically demonstrated along with its limitation.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Genetic Programming*

General Terms

Algorithms, Theory

Keywords

Tree Semantics, Module Acquisition, Schema Theory

1. INTRODUCTION

Unlike GA's linear string chromosome, a GP tree is a set of tightly linked nodes with many inherent complexities. Two prominent ones are *sub-tree context* and *tree fragment evaluation*. Each sub-tree of a GP tree is tightly bound to its parent node and its meaning is dictated by this node. For this reason it is impossible to evaluate a sub-tree independent of its parent node. If the parent node of a sub-tree is changed (change of context) then the meaning of the sub-tree may also change. Look at Fig 1(top) for an example.

GECCO'05, June 25–29, 2005, Washington, DC, USA.

This paper presents a new approach to *explicitly* look at the fitness contributions of the schemata within a tree. Currently, it is a two step offline process. In the first step, a schema is selected from the final generation and compared with the entire populations of all the preceding generations. Every time an individual matches the schema, its fitness is noted. Then, the sub-tree instantiating this schema is replaced by an *intron* and the individual is re-evaluated. The difference between the two fitness values gives an idea of the schema's contribution in the original individual.

This approach is tested on the problems of *module acquisition* and *dead code identification* from GP literature to demonstrate its usefulness. Its possible use in the existing approaches is also discussed briefly.

The paper is organized as follows, the next section gives a brief overview of the existing schema theories. Section 3 discusses our approach in detail followed by detail description of the experiments conducted. Section 5 discusses its possible use in different GP problems. Section 6 concludes the paper by summarizing our findings and our future course of action.

2. EXISTING APPROACHES

After the introduction of GP, there have been a number of attempts to develop a schema theory to help enable us to analyze it in detail. Like GA, most GP schema work involves the identification and analysis of sub-trees and tree fragments. A dont-care(#) is used to *generalize* schema. Depending on the definition of a schema this # can match to any sub-tree [3], rooted sub-tree [6] or to a terminal or a non-terminal [4][5].

The schema theories defined for GP can be divided into two broad categories, *structural approaches* and *functional approaches*. Structural approaches focus mainly on the structure and preservation of these structures in the subsequent generations, whereas functional approaches do consider the fitness of the schema to *some* extent.

2.1 Structural Approach

The most famous structural schema theories were presented by Koza [2], O'Reily and Oppacher [3] and Whigham. According to Koza's definition, a schema is a similarity template for tree structures and represents a subspace that comprises of all the trees that match it. According to him, a schema H represents a set of sub-trees. For example the schema $H = [(* \ x \ y), (- \ x \ 1)]$ represents all the programs having at least one instance of each of $(* \ x \ y)$ and $(- \ x \ 1)$. O'Reilly and Oppacher introduced don't-care(#) nodes in

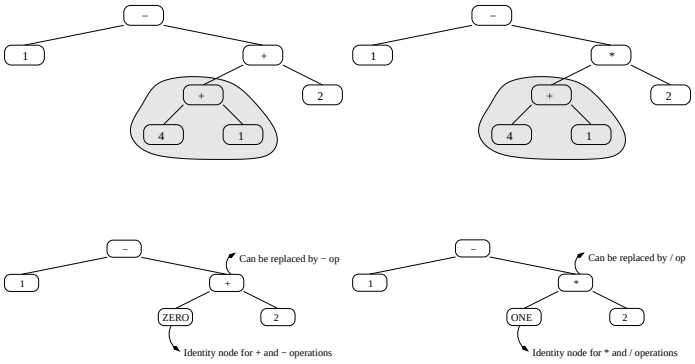

Figure 1: *TOP:* **In this example each GP tree has an instance of the schema (+ # 1) (shown shaded). The left tree evaluates to -6 while the right tree evaluates to -9. To calculate the contribution of the selected sub-trees, each will be replaced by the corresponding "identity" function.** *BOTTOM:* **A sub-tree's contribution is calculated by replacing a sub-tree with an identity node. In the left, sub-tree is replaced with a "ZERO" node due to the presence of the + parent node (can also be -). In the right, the sub-tree is replaced with "ONE" node due to the presence of the * parent node (can also be /). After replacement with identity nodes both trees evaluate to -1.**

their definition. They define a schema as a set of sub-trees and *tree fragments*. A tree fragment is a partial tree with at least one #. According to them a # can match to any sub-tree. Whigham's work involved CFG grammars for the derivation of a GP tree. He defined a schema to be a partial derivation tree that can be instantiated by applying all applicable rules from the supplied grammar to the internal nodes of the selected schema.

2.2 Functional Approach

The functional approaches mostly work by detecting useful *blocks (modules)*. After identification these modules are propagated to the subsequent generations to achieve better performance. The prominent theories include Koza's *ADFs* [2], Angeline's *module acquisition* [1] and Rosca's *adaptive representation through learning (ARL)* [7]. ADFs involve the co-evolution of a solution and a locally defined function with fixed signatures that can be invoked from the main program. This has been criticized, in particular by Angeline who introduced a module acquisition approach which deviates from a fixed signature by randomly selecting a block from a tree and collapsing it into a callable subroutine. Rosca and Ballard have criticized random block selection for subroutine construction. They proposed a heuristic based identification of blocks that promise to be salient components in solution improvement.

This represents a focused effort to find the effective blocks. In general, most of the encapsulated functions did not generalize, and so often weren't useful when used by other individuals. This was because the new functions were often used in a different context.

3. OUR APPROACH

In our study a schema is defined as a sub-tree selected from the final generation and fulfilling following criteria:

(i) it should be present in at least half of the population.

(ii) its own depth should at least be equal to a minimum provided depth. As this was an exhaustive search, all legal depths for each tree were examined.

A schema's presence in 50% of the population is a good measure to estimate its significance. Candidate schemata were searched exhaustively at all the depths within the container-trees of a population. By specifying the size of a schema the effect of a schema with different depth(s), hence size(s) can be studied. Fot this study the selected schema is a complete tree with its root node at a provided depth within a container-tree.

3.1 Schema Generalization

After the selection of a schema, its nodes are replaced probabilistically with # nodes with bias towards the lower nodes. To replace a node, a roulette wheel was constructed. It helps to mark a depth and then a node for replacement. Once a depth is selected, the probability of selection of a node from this depth is uniformly distributed. A # node can match any sub-tree. In the generalization step all the constants in the selected schema are replaced with #. This avoids match failures due to different constants (which is a common observation).

3.2 Schema Contribution

The contextual analysis in this paper is made possible by calculating the fitness contribution of any sub-tree within a tree. This is a three step process. In the first step, an individual containing the sub-tree is evaluated, before the sub-tree is replaced with an *identity* node. The identity node acts as an intron in the container-tree and cancels the effect of the sub-tree. After replacement, the individual is then re-evaluated. The difference of the two fitnesses is the contribution of the sub-tree in the container-tree (see Fig. 1 where, for simplicity, the fitness of an individual is the value it returns). In Fig. 1(top) the left and right trees evaluate to -6 and -9 respectively. In Fig. 1(bottom) the replaced trees evaluate to value -1. Using the aforementioned definition, the fitness contribution of left sub-tree will be $-6 - (-1) = -5$, whereas the contribution of the right sub-tree will be $-9 - (-1) = -8$. This clearly shows the effect of the context on the fitness values.

The working principle of an identity node is similar to the identity function in set theory. In our definition, an

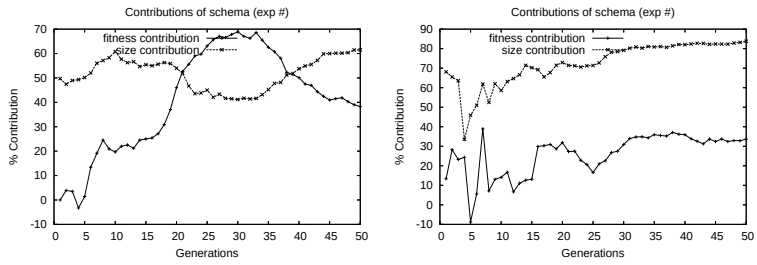

Figure 2: Fitness and size contribution plots of the schema (exp #) for two independent runs.

identity node always replaces some node or sub-tree. This replacement cancels out the effect of the replaced node or sub-tree. For example, in Fig. 1(bottom) the nodes "ZERO" and "ONE" are identity nodes for addition (or subtraction) and multiplication (or division) respectively. The parent nodes of these identity nodes will always return their other argument. [1]

3.3 Limitation of the Proposed Approach

As identity nodes do not exist for all the functions, and only the binary functions (+, -, *, /) have identity functions, hence identity nodes. Due to non-availability of the identity nodes for unary operators, it is impossible to apply this method to the nodes involving unary operators.

4. EXPERIMENTS CONDUCTED

To test our method, we conducted a series of experiments using Koza's quartic polynomial symbolic regression problem, and a population of size 500 was allowed to evolve for 50 generations. Sub-tree crossover with probability 0.9 and reproduction with probability 0.1 were used. The initial generation was generated using the *ramped half and half* method. The initial tree depth varied from 2-6 while the maximum depth of the trees was set to 17, and 100 independent runs were conducted to note the trend of selected schemata across different runs. The same function set as that employed by Koza was used.

We examined three different measurements for this study. The first was to note the **fitness contribution** of the selected schema in a run. This was calculated for each generation by averaging the contribution of all instances of a schema in that particular generation. Similarly, the **size contribution** of each of the identified schema was noted, and averaged in the same way, while the third measurement, **schema depth**, tracked the average depth of each schema.

Due to space limitation we will be showing the result for only first two experiments.

All the schemata fulfilling the selection criteria laid above were selected from the final generation and compared against all other generations of a run. In case of successful match fitness contribution , $f_{contrib}$, of each schema was calculated by equation 1:

$$f_{contrib} = (f_{inc} - f_{exc})/f_{inc} * 100 \quad (1)$$

where f_{inc} and f_{exc} are fitnesses of the schema container-tree with and without schema respectively.

[1]Note, - and / are special cases of + and * respectively. As $a - b = a + (-b)$ and $a / b = a * (1 / b)$.

It could be argued that any increase or decrease in fitness could simply be the result of the removal of a particularly large or small part of the container-tree. To study that, the second experiment looked at the percentage size contribution $size_{contrib}$ of a schema in the overall size of a tree. Let $size_{inc}$ be the size of the tree including the schema and $size_{exc}$ be the size excluding it. The $size_{contrib}$ can then be calculated using equation 2.

$$size_{contrib} = (size_{inc} - size_{exc})/size_{inc} * 100 \quad (2)$$

Note all these tests were conducted *offline* and in a retrospective manner. These results can not be averaged over runs because each run is an *independent* event and may explore different part of the search space. Therefore, it is possible for a schema to perform well in one run but poorly in others. In one typical run, an instance count of a schema varies from a few hundred(initial generations) to several thousands (final generations) in a population of size 500, making our data sample statistically significant.

4.1 Schema contribution to container-tree

Fig 2 shows size and fitness contribution plots of the schema (exp #) for two independent runs. Note the very same schema behaves totally differently in two different runs of a same problem. In the left figure, the size and fitness contributions of the schema are, in general, inversely proportional to each other contrary to its behavior in the second run. This confirms our previously laid hypothesis that each run is an independent event. It can also inferred from the graphs that for the left run the schema is acting significantly in the initial generations, as is contributing 70% to the overall fitness of the container-trees. In the latter part of the run discovery of more significant schema(ta) made it less significant and is evident from the fitness contribution drop of the schema after generation 33. Whereas in Fig 2(right) the significance of the schema, in general, remained constant through out the run.

In Fig. 3 all the schemata found in two independent runs are plotted and compared. These runs are selected as they have schemata of varying size and content (operators, terminals) and are representative of others. In the left graph, all the schemata are behaving similarly, i.e. their contributions drop with time. The schema (/ (/ X #) #) is an exception. Its contribution remains zero till 32nd generation. After this it has started contributing as much as 25% of the fitnesses of the container-trees. Later, we have discovered that the *best-of-run* individual for this run was found in the 32nd generation. In Fig. 3 (right) the longer schemata fail to have any instances in the initial generations but, once found, their

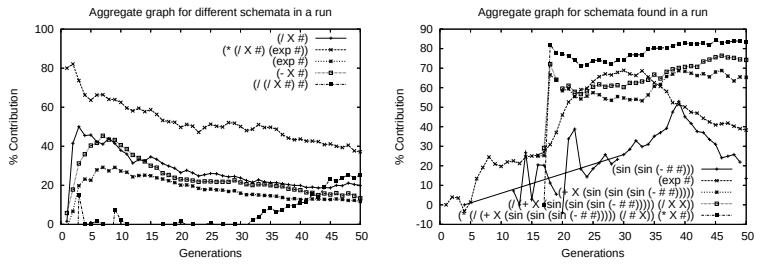

Figure 3: Comparisons of all the schemata found in two independent runs.

Count	Schema
165819	(* # #)
22401	(* X #)
20357	(+ X X)
8256	(* X (exp #))
6876	(+ # #)
6098	(- X X)
6067	(* (exp X) #)

Table 1: Average instance frequency counts of potentially useful schemata that could be encapsulated.

fitness contributions either remain constant or increase with time. Notice the fitness contributions of schemata can be *negative* (look at Fig. 3(right). A schema has a negative fitness contribution when its removal results in an *increase* of fitness of the container-tree.

4.2 Identification of Modules

It is reasonable to assume that a schema significant in *all* the runs (high instance count and fitness contributions) can be identified as a candidate schema for encapsulation. To study this instance counts of potentially useful schemata are noted and are shown in table 1. For example, by consulting Fig. 2(left) and table 1 the schema (*exp* #) can be marked as a useful candidate for encapsulation.

5. BENEFIT OF THIS STUDY

This study has possible use in the following areas:

- Identification of semantically significant schemata from a population during online analysis. These modules can then be used to generate better individuals in the subsequent generations.
- Improvement in selection methods of the existing theories. The proposed approach is more informed method for selection of modules for encapsulation.
- Identification of dead code. Iba and de Garis also calculate the worth of the constituent sub-trees of an individual by treating each sub-tree as an independent program. This evaluation ignores the context provided by the container-tree and, hence, does not inform us about the worth of a sub-tree towards the main tree.
- Definition of *effective crossover* for GP. This can be accomplished by not selecting the nodes involving dead codes as crossover points.

6. CONCLUSION AND FUTURE WORK

We have proposed a new methodology to evaluate a GP schema in context and have demonstrated its use in different GP problems. Its possible use in improving existing schema theories is also mentioned.

In future we are planning to employ it in module identification in online GP system along with its application on different problems.

7. REFERENCES

[1] Peter John Angeline. Genetic programming and emergent intelligence. In Kenneth E. Kinnear, Jr., editor, *Advances in Genetic Programming*, chapter 4, pages 75–98. MIT Press, 1994.

[2] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[3] Una-May O'Reilly and Franz Oppacher. The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73–88, Estes Park, Colorado, USA, 31 July–2 August 1994 1995. Morgan Kaufmann.

[4] Riccardo Poli and W. B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. Technical Report CSRP-97-3, School of Computer Science, The University of Birmingham, B15 2TT, UK, January 1997. Presented at GP-97.

[5] Riccardo Poli and William B. Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[6] Justinian P. Rosca. Analysis of complexity drift in genetic programming. In John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 286–294, Stanford University, CA, USA, 13-16 July 1997. Morgan Kaufmann.

[7] Justinian P. Rosca and Dana H. Ballard. Discovery of subroutines in genetic programming. In Peter J. Angeline and K. E. Kinnear, Jr., editors, *Advances in Genetic Programming 2*, chapter 9, pages 177–202. MIT Press, Cambridge, MA, USA, 1996.