Experiments with Indexed FOR-Loops in Genetic Programming

Gayan Wijesinghe and Vic Ciesielski
School of Computer Science and Information Technology, RMIT University
GPO BOX 2476V, Melbourne, Victoria, Australia
{gayan.wijesinghe,vic.ciesielski}@rmit.edu.au

ABSTRACT

We investigated how indexed FOR-loops, such as the ones found in procedural programming languages, can be implemented in genetic programming. We use them to train programs that learn the repeating unit string of a given regular binary pattern string and can reproduce the learnt pattern to an arbitrary size, specified by a parameter N. We discovered that this particular problem, where the solution needs to scale with multiple size-instances of the problem, is very hard to solve without the help of domain knowledge.

Categories and Subject Descriptors: I.2.2 Artificial Intelligence: Automatic Programming

General Terms: Algorithms

Keywords: genetic programming, machine learning, philosophical aspects of evolutionary computing, representations, theory

Summary

Loops are rarely used in genetic programming due to issues relating to their representation within the program structures and maintaining semantically valid loops throughout the evolutionary process [1, 2]. In this work, we present two ways to represent and use indexed FOR-loops within genetic programming. We apply these representations on learning the repeating unit of a regular binary pattern to obtain programs that can reproduce these patterns to an arbitrary size.

Our research questions in particular are as follows:

1. How can we represent indexed FOR-loops, similar to the ones found in procedural programming, in tree-based genetic programming, while ensuring their structural validity and termination?

2. Can we use the above indexed FOR-loops to learn the repeating unit of a binary pattern so that the pattern can be reproduced to an arbitrary size?

3. How can we improve the representation of these procedural programming-based indexed FOR-loops to better suit the searching characteristics of genetic programming?

We have had some success in evolving program trees with indexed loops for this challenging class of problems. Some of the evolved programs are short, elegant and understandable. However, in order to get success it was necessary to use quite specific domain knowledge in the GP representation.

To answer our 1st research question, we began with the following, more general, representation of FOR loops in GP:

LOOP (TIMES BODY)

In our work, an indexed FOR-loop in GP has a subtree for the number of loop iterations (TIMES) which is a function of the size of the problem and another subtree for the loop body statements (BODY), in general. Allowing an index variable inside this body can result in the body statements scaling accordingly to different sizes of the problem being solved.

To answer our 2nd research question, we followed the above in designing Method 1 for learning regular binary patterns with repeating unit lengths of 2, 4, 6, 8 and 10. Method 1 of Figure 1 shows the exact grammar that we used in this part of our work. In these grammars, N is the problem length. We use $ADD1$ to add 1 to the value at the array position POS when reproducing the strings. All the elements of the arrays were initialised to 0.

As the programs need to learn how to repeat the learnt pattern at various lengths, we trained the programs on examples of multiple lengths, starting from the length of 1 repeating unit up to 10 repetitions of it at consecutive interger intervals. The fitness calculation was made up of two components as we need to check the accuracy of the reproduction as well as the valid operation of the program. For accuracy, we calculated the average character difference between the reproduced and the original patterns, which we refer to as "distance error". The validity and the estimated logical correctness was measured by the "inefficiency" of its average ADD1 statement usage. The final fitness of an individual was formulated such that the selection pressure was on minimising the "distance error" before minimising the "inefficiency". We performed 80 runs per experiment with a GP population size of 100 trees that were limited to a maximum depth of 12 and a maximum generation count of 9000. We used 70%, 28% and 2% for crossover, mutation and elitism rates, respectively.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.
ACM 978-1-60558-130-9/08/07.

Table 1: Comparison of Method 1 and Method 2 Results

Problem	Distance Error		Inefficiency		Total Error		Solution Count	
Length	Method 1	Method 2	Method 1	Method 2	Method 1	Method 2	Method 1	Method 2
2	*0.002045*	0.026859	*0.000750*	0.008639	*0.002795*	0.035498	*79/80*	64/80
4	0.180297	*0.103191*	0.059422	*0.037196*	0.239719	*0.140387*	2/80	*25/80*
6	0.223246	*0.164907*	0.054189	*0.046602*	0.277434	*0.211510*	0/80	*13/80*
8	0.225136	*0.178283*	*0.001980*	0.016065	0.227115	*0.194348*	0/80	*12/80*
10	0.225296	*0.178009*	*0.009279*	0.015430	0.234574	*0.193439*	0/80	*7/80*

Method 1	Method 2
LOOP -> TIMES BODY	LOOP -> TIMES BODY
TIMES -> TIMES / TIMES |	LOOP -> TIMES CENTER BODY
RAND_NUM |	TIMES -> TIMES / TIMES |
N	RAND_NUM |
BODY -> BODY BODY |	N
ADD1	CENTER -> CENTER + CENTER |
ADD1 -> POS	CENTER * CENTER |
POS -> POS + POS |	RAND_CENTER |
POS * POS |	INDEX
RAND_POS |	BODY -> BODY BODY |
INDEX	ADD1_OFFSETx

Figure 1: Grammars for Loops

In Method 1, we obtained 79 solutions (98.75%) for the problem of size 2 and 2 solutions (2.5%) for the problem of size 4 but none for the other problems, as shown in Table 1. In general, we had very limited success with Method 1. However, we observed improvements in fitness over the evolutionary process which indicates that some scalability is possible for the harder problems. The following is a Method 1 solution program found for problem length 4:

```
(LOOP (/ (/ N 2) 2) ((ADD1 (+ INDEX
   (+ INDEX (+ INDEX INDEX)))) (ADD1 (+ INDEX
   (+ INDEX (+ INDEX (+ INDEX 3)))))))
```

By analysing similar solutions of Method 1, we saw how statements within the body need to cooperate with each other in such a way that their relative positioning to each other is not destroyed by evolutionary operations such as crossover and mutation.

In Method 2 we change our representation and grammar to include more domain knowledge (refer Figure 1), such that a loop determines its number of iterations and another function ($CENTER$) that gives the array locations that it will traverse over the iterations. The body statements then perform their own operations relative to these visited locations, which leads to a form of cooperation between them. During crossover and mutation, changes made to $CENTER$ function influence all the body statements equally and when body statements are crossed over and mutated, they still operate relative to the $CENTER$. This method produced solutions for every problem size we tested (as shown in Table 1), with 64 solutions (80%) for the problem of length 2 and 7 solution (8.75%) for the problem of length 10. In general, runs of Method 2 converged faster than that of Method 1, as shown by Figure 2. The following is a Method 2 solution found for problem length 4:

```
(LOOP (/ N 4) (* 4 INDEX)
   (ADD1_OFFSET3 ADD1_OFFSET0))
```

We have also presented a fitness function for problems of this nature that applies pressure for the evolutionary search to find small and efficienct programs. The solutions that we have found in our work demonstrated that they met these fitness criteria.

By the above representations and experiments, we have shown how indexed FOR-loops can be successfully used in tree-based GP for learning the repeating unit of a given regular pattern so that they are able to reproduce the pattern at least to a specific number of arbitrary sizes, sharing similar to human-written programs.

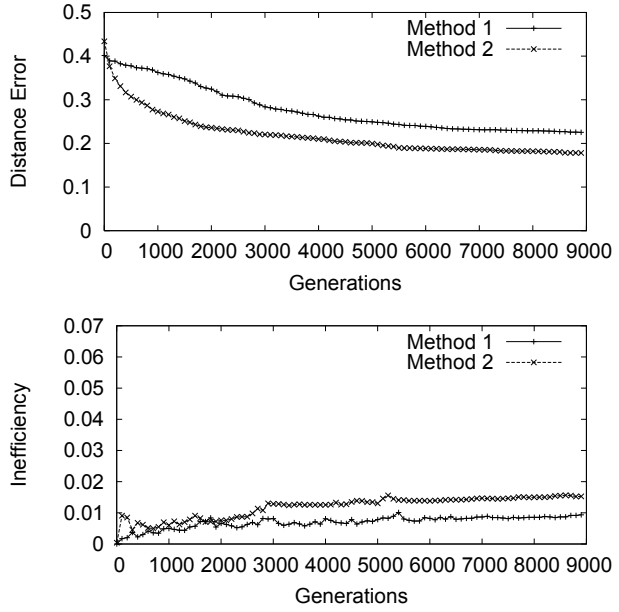

Figure 2: Fitness components of Method 1 compared to Method 2 on problem length 10: Distance error (top), inefficiency (bottom)

1. REFERENCES

[1] V. Ciesielski and X. Li. Experiments with explicit for-loops in genetic programming. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 494–501, Portland, Oregon, 20-23 June 2004. IEEE Press.

[2] G. Wijesinghe and V. Ciesielski. Using restricted loops in genetic programming for image classification. In D. Srinivasan and L. Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 4569–4576, Singapore, 25-28 Sept. 2007. IEEE Computational Intelligence Society, IEEE Press.