Limiting Code Growth to Improve Robustness in Tree-based Genetic Programming

Bart Wyns, Luc Boullart, and Pieter Jan De Smedt
Dept. of Electrical, Systems and Automation, Ghent University
Technologiepark 913
Zwijnaarde, Belgium
Bart.Wyns@UGent.be, Luc.Boullart@UGent.be, Pieter@autoctrl.UGent.be

ABSTRACT

In this paper we analyze the composition of the function set of the artificial ant problem to define new training and testing trails similar to the Santa Fe trail. Cross-validation is used in applications where large amounts of data are available. We also use a semantically driven growth limiter to reduce program size and check if growth reduction could lead to increased test performance.

Categories and Subject Descriptors: I.2.2 [Artificial Intelligence]: Automatic Programming Program synthesis; I.2.6 [Artificial Intelligence]: Learning Concept Learning and Induction; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms: Algorithms, Design, Experimentation

Keywords: representational bias, code growth, robustness, artificial ant, genetic programming

1. INTRODUCTION

Variable length structures start to grow beyond control during evolution, often without clear improvement in best fitness. In genetic programming (GP) this is called code growth [1]. Bloating is an important issue when searching for a general, widely applicable solution. Usually the fitness case used for training program trees is learnt 'by heart'. When evaluating on similar test fitness cases, performance drops dramatically, even if there is only a fine distinction between train and test fitness cases. If GP wants to become more than just another optimization method (GP already has all the necessary tools for evolving robust solutions), more attention should be given to this topic.

2. CREATING SIMILAR ENVIRONMENTS IN THE ARTIFICIAL ANT

The fact that there is only one sensor available to the ant is very important. This sensor can only sense the position on the grid right in front of the ant. Different methods exist to improve this sensor such as 360° view or large distance sensing. But this changes the problem definition, which is to evolve a robust control strategy using only basic primitives. We introduce a different approach to generate new environments for the artificial ant. Instead of analyzing the trail that is used, we take a closer look at the possibilities given by the function set. Based on the characteristics of the IF-FOOD-AHEAD sensor we redefine a building block as follows:

$$\textit{building block} \quad ::= \quad (\square) * \quad (L|R)? \quad \blacksquare$$

where ■ equals food and □ denotes an open space in the trail. A building block is thus defined as a juxtaposition of zero or more empty spaces, optionally followed by a turn and ending with a food pellet. This definition describes the entire class of trails that a generalizing ant should be able to traverse correctly based on the possibilities of the function set.

3. EXPERIMENTING

Four test problems were used to validate the effect of LOSE on best fitness: the artificial ant and three databases from the UCI machine learning repository. A brief description of a semantically driven operator (LOSE) to reduce code growth is given in [2]. We have shown that LOSE provides a significant reduction in program size. Also improved *train* fitness has been observed.

When looking at test fitness, LOSE always significantly increases performance while evolving small and compact solutions. This result is rather surprising because LOSE uses problem specific knowledge to search for a suited subtree (see best subtree selection). This increases the risk of overfitting the training set. But apparently by stimulating smaller sized subtrees (and upgrading them to new individuals) robustness is favored. We also show that GP with LOSE achieves better results with fewer train cases.

4. CONCLUSIONS

In summary, we designed an algorithm for automatically generating trails in the artificial ant based on the function set. We also examined the influence on robustness of a semantically driven growth limiter and showed that LOSE offers an added value over using multiple train cases in all four benchmark problems. LOSE also reduced the number of necessary train cases to achieve robust individuals.

5. REFERENCES

[1] S. Luke. Modification point depth and genome growth in genetic programming. *Evolutionary Computation*, 11(1):67–106, 2003.

[2] B. Wyns, S. Sette, and L. Boullart. Self-improvement to control code growth in genetic programming. *Lecture Notes in Computer Science*, 2936: 256–266, 2004.

GECCO'07, July 7–11, 2007, London, England, United Kingdom.
ACM 978-1-59593-697-4/07/0007.