An Investigation into Using Genetic Programming as a Means of Inducing Solutions to Novice Procedural Programming Problems

Nelishia Pillay
School of Computer Science
Pietermaritzburg Campus,
University of KwaZulu-Natal
KwaZulu-Natal, South Africa
Tel: +27 33 2605644
E-mail: pillayn32@ukzn.ac.za

ABSTRACT

The study presented in this paper forms part of a larger initiative aimed at creating a generic architecture for the development of intelligent programming tutors (IPTs) in an attempt to reduce the costs associated with building IPTs. Thus, instead of requiring the lecturer to provide solution algorithms to the programming problems that students will be tested on by the system, the generic architecture will automatically generate the solutions to these problems. This paper reports on the results of an investigation conducted to test the hypothesis that genetic programming (GP) can be used for this purpose. The paper proposes a genetic programming system for the induction of solutions to arithmetic, character and string manipulation, conditional, iterative, nested iteration, and recursive problems. The paper analyses the results of applying the proposed system to 45 randomly chosen novice procedural programming problems. Extensions made to the proposed system based on this analysis, namely, the implementation of the iterative structure-based algorithm (ISBA), are discussed.

Categories and Subject Descriptors

I.2.2 [Artificial Intelligence]: Automatic Programming – *program synthesis.*

General Terms: Algorithms, Experimentation, Theory.

Keywords

Genetic programming, automatic programming, local optima.

1. INTRODUCTION

One of the functions of an intelligent programming tutor (IPT) is to assess a learner's programming skills by comparing solution algorithms or programs written by a student to the stored solutions written by the developer of the IPT. In order to cater for more than one solution algorithm for a programming problem, alternative solutions to a problem are stored in the knowledgebase.

GECCO'05, June 25-29, 2005, Washington, DC, USA.
ACM 1-59593-010-8/05/0006.

Thus, in order to reduce the developmental costs of IPTs, one of the functions of the Expert Module [3] of the generic architecture is the automatic induction of the solution algorithms usually stored in an IPT knowledgebase. The study presented in this paper evaluates genetic programming as a means of evolving solution algorithms to novice procedural programming problems.

This study focuses on the following introductory programming concepts, namely, arithmetic, character and string manipulation, conditional control structures, iteration, nested iteration and recursion. Furthermore, the study is delimited to evaluating genetic programming as a means of finding at least one solution to each problem. Future extensions of the project will examine evolving multiple solution algorithms, each taking a different approach to solving the same problem. It is evident from the literature that the algorithms generated by GP systems usually contain redundant code called introns. The removal of introns will also be investigated at a later stage.

2. PROPOSED GP SYSTEM

An internal representation language was used to express algorithms. The elements of this language are listed in **Table 2.1**. Input to the genetic programming system is a problem specification describing the problem input and output, the set of fitness cases containing values for each input and corresponding output, the application domain, screen output in the case of ASCII graphics problems and a subset of the internal representation language that students should have knowledge of to solve the particular problem.

In order to ensure that the programs generated have a legal structure and to facilitate the translation of the evolved algorithms into a particular programming language, the system is strongly-typed. Each terminal, constant, memory location, function and function argument is of a specific type. The types catered for by the system are Integer, Real, Boolean, Char, String, Output, and Generic (can be any of the types listed).

An individual is evaluated by executing the individual on each set of input values described in the fitness cases and comparing the output of the execution with the target output specified in the fitness cases. If the target output is numerical the error fitness function defined by Koza [2] is used to calculate the fitness of the individual.

For all other target output types the fitness measure is the number of fitness cases for which the individual produces exactly the same output as the target output. In the case of ASCII graphics problems the output written to the screen maintained by the system is compared to that described in the problem specification. The individual is penalized if a screen location is written to more than once or an attempt has been made to access a location beyond the bounds of the screen. The tournament selection method is used to choose parents of the successive generations.

Table 2.1: Internal Representation Language

Type	Operators
Arithmetic	+, -, *, /, sqrt, %, neg, sq, cube, pow, trunc, round, ceil, floor, abs
String	length, concat, concatc , del, delete, insert, copy, equal, charat, upcase
Logical	==, !=, <=, >=, <, >, , cnoteq, bequal bneq, not and, or
Memory	write, change, aread, alen
Conditional	if, switchc, switchi
Iterative	for, while, dowhile
Input & Output	place, toscreen, newline
Multiple Statements	blockn, n=1-5

Lack of genetic diversity, selection variance and destructive genetic operators have been cited as the main causes of premature convergence. In order to promote genetic diversity the reproduction operator is not used and duplicates are not permitted in the initial population. Furthermore, mutation application rates will be increased if necessary. In order to deal with selection variance the system performs multiple iterations per seed, in the hope that a different area of the search space will be visited on each iteration. The system provides non-destructive mutation and crossover operators for use if necessary. These operators produce offspring that are at least as fit as their parents.

The following genetic programming parameters were varied to find a solution for each problem: control model, method of initial population generation, initial tree depth, population size, mutation tree depth limit, tournament size, bound, error offset, type of genetic operators (standard or non-destructive). The following section provides a brief overview of the experimental methodology employed and the results obtained from applying the proposed system to 45 procedural programming problems.

3. RESULTS AND DISCUSSION

The "proof by demonstration" methodology [1], with two iterations and one step of refinement, was employed to test the hypothesis that genetic programming can induce solutions to novice procedural programming problems. If the system was unable to evolve a solution for at least one seed for each problem, changes were made to the system during the refinement process and the system was re-tested. The proposed system was implemented using the Professional version of JBuilder 4 with the JDK 1.3.

The simulations in this study were run on two different systems, namely, a Pentium III with Windows XP and a Pentium 4 with Windows 2000.

The 45 problems used to test the system consisted of 3 sets of ten problems requiring the use of sequential structures, conditional structures, and iterative structures respectively, ten ASCII graphics problems, and five recursion problems. The system was able to successfully evolve solutions to 33 of the problems. For 24 of the 33 problems multiple runs per seed proved to successfully escape local optima caused by selection variance. For 3 of these 33 problems the use of non-destructive operators was needed in order to evolve solutions. A study of the evolutionary process for the 12 problems the system was unable to evolve solutions for revealed that fitness function biases against certain primitives or combinations of primitives resulted in premature convergence.

The iterative structure-based algorithm (ISBA) was developed to escape local optima caused by fitness function biases. The ISBA performs both a global level and a local level search. During each of the runs of the global level search a similarity index is used to ensure that areas of the search space visited on previous runs are not revisited. For each global level run *n* local level runs are performed. On each of the local level runs the structure of each individual is fixed from the root to some level *d* to be the same as the individual converged to on that particular global level run. The revised GP system implementing the ISBA evolved solutions to the 12 problems not solved by the proposed system.

4. CONCLUSIONS AND FUTURE WORK

The study presented in this paper evaluated genetic programming as means of evolving novice procedural solution algorithms. The genetic programming system implemented was able to evolve solutions to the 45 randomly chosen procedural programming problems. The use of multiple runs and non-destructive operators successfully escaped local optima caused by selection variance and destructive operators respectively. The ISBA was developed to escape local optima caused by fitness function biases. Future extensions of the system include the evolution of efficient algorithms, algorithms that adhere to good programming practice, modular programs, object-oriented programs, multiple solutions to each problem and the removal of redundant code.

5. ACKNOWLEDGMENTS

This material is based upon work supported by the National Research Foundation. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and the NRF does not accept any liability in regard thereto.

6. REFERENCES

[1] Johnson, C. *What is Research in Computing Science?*, Department of Computer Science, Glasgow University, http://www.dcs.gla.ac.uk/~johnson/teaching/research_skills/basics.html.

[2] Koza, J.R. *Genetic Programming I: On the Programming of Computers By Means of Natural Selection*, MIT Press, 1992.

[3] Pillay, N. Developing Intelligent Programming Tutors for Novice Programmers. In *inroads-the SIGCSE Bulletin*, 35, 2, ACM Press, June 2003, 78 – 82.