Heuristic Approaches to Program Synthesis: Genetic Programming and Beyond – The Homework Assignment

PhD Open, University of Warsaw, Jan 15-17 2015
Krzysztof Krawiec
Laboratory of Intelligent Decision Support Systems
Institute of Computing Science, Poznan University of Technology, Poznań, Poland

Please choose **one of** the following assignments. The projects should be sent by email to krawiec@cs.put.poznan.pl. The due date is **February 15, 2015**.

1 Assignment 1: Reading

1. Read **one of** the following papers related to program synthesis:
   (a) *Theorems for free!* [Wadler, 1989]
   (b) *Elementary landscape analysis* [Whitley & Sutton, 2009]
   (c) *No free lunch theorems for optimization*, [Wolpert & Macready, 1997] or [Schumacher et al., 2001]
   (d) *A deductive approach to program synthesis* [Manna & Waldinger, 1980]
2. Write a short (up to 5 pages) essay revolving around the paper. The essay should:
   (a) review the main concepts and theses of the paper,
   (b) discuss the consequences of the theses formulated therein,
   (c) identify and discuss the limitations of the approaches/method discussed in the paper,
   (d) where possible, relate to your personal research interest (like, e.g., the possible links to the topic of your PhD).

2 Assignment 2: Software project

The task is to:

1. Step 1: Design a simple domain-specific language (DSL)
2. Step 2: Implement a minimal framework for genetic programming (GP), in the programming language and software environment of your choice
3. Step 3: Demonstrate the operation of the GP framework on a simple program synthesis task expressed in the DSL defined in Step 1

The following steps describe how to achieve this for a simple DSL of arithmetic expressions. But you may pick any other domain of reasonable complexity. The expected outcome is the software and a short report summarizing the DSL, the GP framework, and the results obtained on an exemplary program synthesis task.

2.1 Step 1

Consider the followng stack-based programming language (cf. the Push programming language presented as a part of the course), with the following properties:

1. Instruction set: +, -, *, /. A program is any finite sequence of such instructions.
2. The working memory is a stack of numbers. The program to be executed is expecting the stack to contain the input data. Once a program has terminated, whatever is left on the stack is to be interpreted as program output
3. Each instruction pops the appropriate number of arguments from the stack, performs the operation, and pushes the result on the top of the stack.

Example of program execution:

The program: (- * +)
The initial content of the stack (program input): (3 2 7 4)

Program	Stack
(- * +)	(3 2 7 4)
(* +)	(1 7 4)
(+)	(7 4)
()	(11)

Thus, program output is: (11)

Other design choices to be considered:

1. If there are not enough arguments on the stack, the program should halt.
2. Possible extensions of the instruction set: instruction *dup* (duplicates the top element on the data stack), *pop* (discards the top element from the stack), or a primitive for loop, e.g.: for pops a number n from the stack, and causes the *next* instruction to be executed n times. Consider inventing your own instructions. See Push programming language specification for more examples).

2.2 Step 2

Prepare a minimalistic GP framework based on the DSL. The framework should be able to:

- Maintain a population of programs expressed in the DSL (e.g, strings of arithmetic expressions in the above case),
- Use at least one search operator to produce new candidate programs, e.g.:
  - a mutation operator that replaces a randomly selected instruction in a program with another random instruction,
  - a mutation operator that appends a randomly selected instruction to a program,
  - a crossover operator that slices two parent programs at two random locations and swaps the 'tails',
- Drive the search using a straightforward fitness function, like the number of examples/tests on which the program behaves correctly.

2.3 Step 3

Apply your GP framework and DSL to a simple program synthesis task. For the above arithmetic example, this could be:

- A program that reproduces an output of an arbitrary arithmetic expression
- A program calculating factorial (requires extended instruction set)
- A program finding the smallest element on the stack (requires extended instruction set)

References

[Manna & Waldinger, 1980] Manna, Z. & Waldinger, R. (1980). A deductive approach to program synthesis. *ACM Trans. Program. Lang. Syst.*, 2(1), 90–121.

[Schumacher et al., 2001] Schumacher, C., Vose, M. D., & Whitley, L. D. (2001). The no free lunch and problem description length. In L. Spector & E. D. Goodman (Eds.), *GECCO 2001: Proc. of the Genetic and Evolutionary Computation Conf.* (pp. 565–570). San Francisco: Morgan Kaufmann.

[Wadler, 1989] Wadler, P. (1989). Theorems for free! In *Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, FPCA '89 (pp. 347–359). New York, NY, USA: ACM.

[Whitley & Sutton, 2009] Whitley, L. D. & Sutton, A. M. (2009). Elementary landscape analysis. In *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference* (pp. 3227–3236). New York, NY, USA: ACM.

[Wolpert & Macready, 1997] Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1), 67–82.